Browsing Source Code

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

Part No.: 801-7102-10
Revision A, August 1994

Please
Recycle

Adobe PostScript

Contents

Preface xi

1. Introduction to SourceBrowser 1-3

1.1 How SourceBrowser Works. 1-4

1.2 Graphical Overview 1-5

1.2.1 SourceBrowser Base Window 1-6

1.2.2 CallGrapher 1-12

1.2.3 ClassGrapher 1-14

1.2.4 ClassBrowser 1-17

1.2.5 How the Browsers and Graphers Interrelate 1-19

1.3 Synopsis of SourceBrowser Tasks. 1-21

1.4 SourceBrowser Option Flags. 1-22

2. SourceBrowser Basics 2-25

2.1 Generating a SourceBrowser Database 2-25

2.1.1 Comparison of Database Types. 2-26

2.1.2 Using `make` to Turn on SourceBrowser 2-28

2.1.3 Not Using `make` to Turn on SourceBrowser 2-29
2.1.4 Scanning: Collecting Information without Compilation 2-29
2.1.5 Query/Grep Command 2-31
2.2 Setting up SourceBrowser for Large Applications 2-32
2.3 Starting SourceBrowser 2-34
2.3.1 From the Manager..... 2-34
2.3.2 From a Command or Shell Tool Window..... 2-35
2.3.3 SourceBrowser Base Window 2-35
2.4 Changing the Directory 2-36
2.5 Getting Help 2-38
2.6 Quitting SourceBrowser 2-39
3. Issuing A Simple Query 3-41
3.1 Issuing the First Query 3-41
3.1.1 Query Line..... 3-44
3.1.2 Match Pane 3-44
3.1.3 File Information Field..... 3-45
3.1.4 Source Pane 3-45
3.1.5 Footer 3-45
3.2 Issuing Subsequent Queries 3-46
3.3 Tips on Issuing Queries 3-47
3.4 If Your Query Fails 3-49
4. Viewing Matches 4-51
4.1 Redisplaying the Current Match 4-51

4.2 Viewing the Next Match 4-52
4.3 Viewing the Previous Match 4-54
4.4 Viewing a Selected Match 4-56
4.5 Moving to a Match From Any Query. 4-56
4.6 Moving Between Queries 4-59
4.7 Removing Matches. 4-59
4.8 Removing Queries 4-60
5. Graphing Functions and Classes 5-61
5.1 Graphing Function Calls 5-62
5.1.1 Graphing Overloaded and Virtual Functions in C++ Programs. 5-64
5.1.2 Expanding Functions 5-67
5.2 Graphing Class Inheritance. 5-70
5.2.1 Expanding Classes 5-72
5.2.2 Browsing Classes. 5-75
5.3 Adding a Node to the Graph 5-76
5.4 Finding a Node in the Graph 5-78
5.5 Editing the Graph. 5-79
5.6 Redoing the Layout 5-81
5.7 Printing the Graph 5-81
5.8 Querying a Function or Class 5-81
5.9 Displaying the Source of a Function or Class 5-83
5.9.1 Displaying Source in SourceBrowser 5-84
5.10 Customizing the Call and Class Graphs 5-85

5.11 Quitting the CallGrapher and ClassGrapher 5-92
6. Browsing Classes 6-93
6.1 Browsing the First Class 6-94
6.1.1 Activating ClassBrowser From SourceBrowser.. 6-94
6.1.2 Activating ClassBrowser From ClassGrapher... 6-96
6.2 Browsing Subsequent Classes..... 6-97
6.2.1 In the Existing ClassBrowser Window..... 6-98
6.2.2 In a New ClassBrowser Window 6-98
6.3 Viewing Data and Function Members..... 6-101
6.4 Viewing Virtual and Inline Functions 6-103
6.5 Traversing the Class Hierarchy..... 6-104
6.6 Revisiting Previously Browsed Classes..... 6-105
6.7 Graphing Class Inheritance..... 6-106
6.7.1 Manually Graphing a Class..... 6-106
6.7.2 Automatically Graphing a Class..... 6-106
6.8 Listing Class Types..... 6-107
6.9 Querying for Classes and Functions 6-110
6.10 Displaying the Source of Classes and Functions 6-112
6.10.1 Displaying Source in SourceBrowser 6-112
6.11 Customizing ClassBrowser..... 6-114
6.12 Quitting ClassBrowser..... 6-116
7. Restricting the Match Set 7-119
7.1 Filtering a Query 7-119
7.1.1 Setting the Filter 7-120

7.1.2 Resetting the Filter 7-123
7.1.3 Clearing the Filter 7-124
7.1.4 Filtering to Reduce the Number of Matches 7-124
7.1.5 Filtering in Macro Definitions 7-126
7.1.6 Filtering Different Uses of the Same Symbol. . . . 7-128
7.1.7 Filtering to Obtain Very Specific Matches 7-128
7.2 Focusing A Query. 7-129
7.2.1 Setting the Focus 7-130
7.2.2 Resetting the Focus Window 7-135
7.2.3 Updating the Focus Window. 7-135
7.2.4 Clearing the Focus 7-136
7.2.5 Focusing on Selected Files 7-136
7.2.6 Focusing on More than One Unit of Code 7-137
8. Integrating Editing and Debugging 8-139
8.1 Editing Source Code. 8-139
8.2 Using SourceBrowser with the Debugger 8-141
8.2.1 Displaying Source for CallGrapher 8-145
8.2.2 Displaying Source for ClassBrowser 8-147
8.3 Using an External Editor with the SourceBrowser and the Debugger 8-150
8.3.1 Starting the Tools from the External Editor 8-151
8.3.2 The Debugger and the External Editor. 8-152
8.3.3 The SourceBrowser and the External Editor 8-152
9. Issuing More Advanced Queries 9-153

9.1 Querying for String Constants 9-153
9.2 Using Wildcards in Queries 9-154
9.2.1 Shell Style Patterns 9-155
9.2.2 Regular Expressions 9-156
9.2.3 Literal Strings 9-158
9.3 Using Strings and Wildcards Together 9-158
9.4 Issuing Case-Insensitive Queries 9-159
10. Customizing SourceBrowser. 10-161
10.1 Setting SourceBrowser Properties 10-162
10.2 Understanding SourceBrowser Properties 10-163
11. Overview of the Database 11-171
11.1 Creating and Updating the Database. 11-171
11.2 Linking Executable Files When You Build the Database 11-177
12. Working with Multiple Directories 12-179
12.1 Installing a Symbolic Link between Directories 12-179
12.2 Consolidating Several Databases 12-180
12.3 Using sbinit to Browse Multiple Directories. 12-182
12.3.1 Reading Databases in Other Directories 12-183
12.3.2 Writing to Databases in Other Directories 12-184
12.3.3 Browsing on Multiple Machines 12-186
12.3.4 Using `replacepath` 12-187
12.3.5 If You Cannot Find the .sbinit File 12-188
13. Controlling the Database. 13-189
13.1 Manually Updating the Database. 13-189

13.2 Accessing a Locked Database 13-190

A. Troubleshooting A-195

A.1 Problem Checklist. A-195

A.2 Property Default Values. A-198

A.3 Reporting Problems A-200

A.4 SourceBrowser Messages. A-201

A.4.1 CallGrapher and ClassGrapher. A-207

A.4.2 ClassBrowser A-209

B. Issuing a Command-line Query B-213

B.1 Issuing a Query. B-213

B.2 Command-line Options B-214

B.2.1 `-pattern` <*symbol*> B-215

B.2.2 `-break_lock` B-216

B.2.3 `-files_only` B-216

B.2.4 `-help` B-217

B.2.5 `-help_focus` B-217

B.2.6 `-help_filter` B-218

B.2.7 `-max_memory` <size> B-220

B.2.8 `-no_case` B-220

B.2.9 `-no_secondaries` B-221

B.2.10 `-no_source` B-221

B.2.11 `-no_update` B-222

B.2.12 `-o` <file> B-223

B.2.13 `-show_db_dirs` B-223

B.2.14 -symbols_only B-224
B.2.15 -version B-224
B.2.16 -sh_pattern B-225
B.2.17 -reg_expr B-225
B.2.18 -literal B-226
C. Integration with CASE Tools C-227
C.1 Introduction C-227
C.2 CASE Tool Start Up C-228
C.3 Errors C-228
C.4 Synchronized Browsing C-229
C.4.1 SPRO_SB_BROWSE_TOOL Message C-231
C.4.2 SPRO_Visit_Object Message C-232
C.4.3 SPRO_Browse_File Message C-233
C.5 Reverse Engineering C-234
C.5.1 SPRO_SB_GEN_IDF Message C-234
C.5.2 SPRO_SB_SYNC_IDF Message C-235
C.5.3 SPRO_SB_IDF_GENERATED Message C-236
C.5.4 IDF File Format C-237
Glossary Glossary-245
Index Index-253

Preface

This manual explains how to use the SourceBrowser, a code browser for software application developers. SourceBrowser is an integrated component of the SPARCworks/ProWorks toolset, which also includes:

- Analyzer
- Debugger
- FileMerge
- MakeTool
- Manager

Before You Begin

This manual is written for application developers who want to use SourceBrowser on Sun® workstations to aid in development using ANSI C, C++, FORTRAN, Pascal, and Assembler languages.

This manual assumes you are familiar with

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

  If you are not familiar with the OPEN LOOK interface, see Chapter 4, "The OPEN LOOK GUI," in *Managing the Toolset*.

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

Solaris 2.1 and 1.X

From a usage point of view, almost all of the aspects of the SPARCworks and ProWorks Manager (under Solaris 2.1 and 1.X) are the same. This includes functionality, behavior, usage, and features. The few details that are different, are described in the documentation.

Operating Environment

The ProWorks and SPARCworks toolsets run under the Solaris™ 2.x and 1.x operating environments.

For ProWorks, Solaris™ 2.x for the x86 environment implies:

- Solaris 2.1 (or later) operating environment
- SunOS™ 5.1 (or later) operating system
- An x86 computer (either a server or a workstation)
- The OpenWindows 3.x application development platform

For SPARCworks, Solaris 2.x implies:

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

For SPARCworks, Solaris 1.x implies:

- Solaris 1.2 (or later) operating environment
- SunOS 4.1.X operating system
- A SPARC computer (either a server or a workstation)
- The OpenWindows 3.x application development platform

The SunOS 5.0 operating environment is based on the System V Release 4 (SVR4) UNIX[1] operating system, and the ONC™ family of published networking protocols and distributed services. SunOS 4.1.X is based on the UCB BSD 4.3 operating system.

Compatibility

The compilers and SourceBrowser you are using must be from the same release level. SourceBrowser Version 3.0, and later, must be used with SPARCcompilers 3.0, and later. Some incidental incompatibilty exists. Release 3.0 and 3.0.1 are compatible.

The SPARC and x86 versions of the SourceBrowser are incompatible. A SourceBrowser databse generated on x86 cannot be used with a SPARC SourceBrowser, and vice versa. Thus, you may not intermix databases from these architectures. The incompatilbilty may cause a segmentation violation, which in releases prior to 3.0.1, will not be detected.

How This Book Is Organized

This manual is organized as follows:

Part 1—Browsing Overview

This section introduces the structure and fundamental terminology behind SourceBrowser.

Chapter 1, "Introduction to SourceBrowser," provides you a graphical overview of the four SourceBrowser windows and shows you how they interrelate.

Part 2—Basic Browsing

This section describes step-by-step how to use SourceBrowser, beginning with simpler concepts and moving toward the more complex.

Chapter 2, "SourceBrowser Basics," explains how to start SourceBrowser in a window environment.

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

Chapter 3, "Issuing A Simple Query," describes how to issue your first and subsequent queries for identifiers.

Chapter 4, "Viewing Matches," describes how you can view matches with their surrounding source code. A match is a symbol for which you have queried.

Chapter 5, "Graphing Functions and Classes," describes how you graph functions in supported languages and classes in C++ programs.

Chapter 6, "Browsing Classes," describes how you browse through C++ classes, code, and libraries quickly and easily.

Chapter 7, "Restricting the Match Set," describes how to limit your search for symbols based on how they are used in a program. It also describes how to focus your search on instances of specific units of code.

Chapter 8, "Integrating Editing and Debugging," describes how you can examine source code in order to detect areas of your program you may need to change. It also describes using an external editor as well as intermixing the browsing, editing, and debugging features.

Chapter 9, "Issuing More Advanced Queries," describes how you search for string constants and search patterns that contain wildcards. It also describes how you use command-line options in queries.

Chapter 10, "Customizing SourceBrowser," describes how to customize SourceBrowser to suit your specific needs.

Part 3—Advanced Browsing

This section gives insight into improving SourceBrowser performance.

Chapter 11, "Overview of the Database," describes how SourceBrowser creates and updates its database.

Chapter 12, "Working with Multiple Directories," describes how you work with source files whose database information is stored in multiple directories.

Chapter 13, "Controlling the Database," describes the commands you use to build and update the database.

Appendixes

Appendix A, "Troubleshooting," lists the SourceBrowser messages and offers you instructions about what to do next.

Appendix B, "Issuing a Command-line Query," describes how you browse your source code from the command-line.

Appendix C, "Integration with CASE Tools," describes the SourceBrowser support of integration with CASE tools.

Glossary

The **Glossary** describes terms specific to SourceBrowser.

What Typographic Changes and Symbols Mean

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

Table P-1 Typographic Conventions

Typeface or Symbol	Meaning	Example
`AaBbCc123`	The names of commands, files, and directories; on-screen computer output	Edit your `.login` file. Use `ls -a` to list all files. system% You have mail.
`AaBbCc123`	What you type, contrasted with on-screen computer output	`system% su` `Password:`
AaBbCc123	Command-line placeholder: replace with a real name or value	To delete a file, type `rm` *filename*.
AaBbCc123	Book titles, new words or terms, or words to be emphasized	Read Chapter 6 in *User's Guide*. These are called *class* options. You *must* be root to do this.
♦	A single-step procedure	♦ **Click on the Apply button.**

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

Table P-1 Typographic Conventions *(Continued)*

Typeface or Symbol	Meaning	Example
%	C shell prompt	`system%`
$	Bourne and Korn shell prompt	`system$`
#	Superuser prompt, all shells	`system#`

How to Get Help

SPARCworks/ProWorks tools include the following on-line help facilities:

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

  To start the AnswerBook system, type: `answerbook`

- **Magnify Help™** messages are a standard feature of the OpenWindows software environment. If you have a question, place the pointer on the window, menu, or menu button and press the Help key.

- **Notices** are a standard feature of OPEN LOOK. Some notices ask whether or not you want to continue with an action. Others provide information about the end result of an action and appear only if the end result of the action is irreversible.

- **SunOS Manual Pages** (man pages) provide information about the command-line utilities of the SunOS operating system. Each tool has at least one man page.

  The SourceBrowser manual pages include:
  - `sbtags`(1)
  - `sbrowser`(1)
  - `sbquery`(1)
  - `sbinit`(4)
  - `sbcleanup`(1)
  - `sparcworks`(1) (for SPARCworks)
  - `proworks`(1) (for ProWorks)

♦ **To access the man pages, type**: `man` *utility_name*

Related Documentation

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

- *Installing SunSoft Software on Solaris* (for SPARCworks)
- *SPARCworks/ProWorks Tutorial*
- *Building Programs with MakeTool*
- *Debugging a Program*
- *Managing the Toolset*
- *Merging Source Files*
- *Performance Tuning an Application*

Part 1—Overview of SourceBrowser

Introduction to SourceBrowser 1

This chapter describes SourceBrowser, its components, and its graphical user interface (GUI).

This chapter is organized into the following sections:

How SourceBrowser Works	*page 1-4*
Graphical Overview	*page 1-5*
Synopsis of SourceBrowser Tasks	*page 1-21*
SourceBrowser Option Flags	*page 1-22*

SourceBrowser is an interactive tool that aids programmers in the development and maintenance of software systems, particularly large ones.

During the course of a programming project, you may join a programming team to enhance, maintain, or port code. However, before you become a productive team member, you must fully understand the code that you will modify.

SourceBrowser is a powerful tool for helping you quickly browse large programs. Specifically, SourceBrowser assists you by finding *all* occurrences of any symbol or string of your choice, including those found in header files.

SourceBrowser uses a “What you see is what you browse” paradigm. The source code you edit and compile is the same source code SourceBrowser uses in its searches.

1

SourceBrowser is an open system that has been designed to be used with multiple languages. Currently, SourceBrowser can be used with C++, ANSI C, FORTRAN, Pascal, and SPARC Assembler.

When you browse a program which is written in more than one language, SourceBrowser automatically determines the language in which each source file is written. The browsing operations do not change from one language to another.

You can use SourceBrowser in either the OpenWindows or command-line environment. While the SourceBrowser window environment can be accessed only from a Sun workstation, the command-line environment can be used from terminals and from workstations emulating terminals.

An editor option exists with which you can use an external editor (if it uses the ToolTalk protocol) to replace the SourceBrowser's source displays. Use of an external editor is described in this chapter under, **"***SourceBrowser Option Flags*,**"**. Full details are given in Chapter 8, **"***Integrating Editing and Debugging*,**"**.

1.1 How SourceBrowser Works

You use SourceBrowser by issuing queries. A query instructs SourceBrowser to find all occurrences of the symbol, string constant, or search pattern that you have specified. You then view matches (occurrences of the item you requested SourceBrowser to find) with their surrounding source code.

You can also graph the function relationships in your program. If your source code is written in C++, then you can browse and graph the classes defined in your program.

SourceBrowser responds to queries by searching in a database that contains information about the files you are are browsing. You create this database when you compile your source file with the SourceBrowser option.

The database consists of two parts:

- One file with an extension of `.bd` (browser data) for each file for which you have generated SourceBrowser information.
- An index file which SourceBrowser uses to quickly locate information in the `.bd` files. SourceBrowser creates this file when you issue your first query.

Once SourceBrowser has built its database and index, it can operate efficiently even on very large programs.

If you cannot compile or do not wish to compile, you can use either the scanning procedures described in Chapter 2 under, **"***Collecting Information without Compilation*,**"** or the `grep` command as described in Chapter 2 under, **"***Query/Grep Command*,**"**.

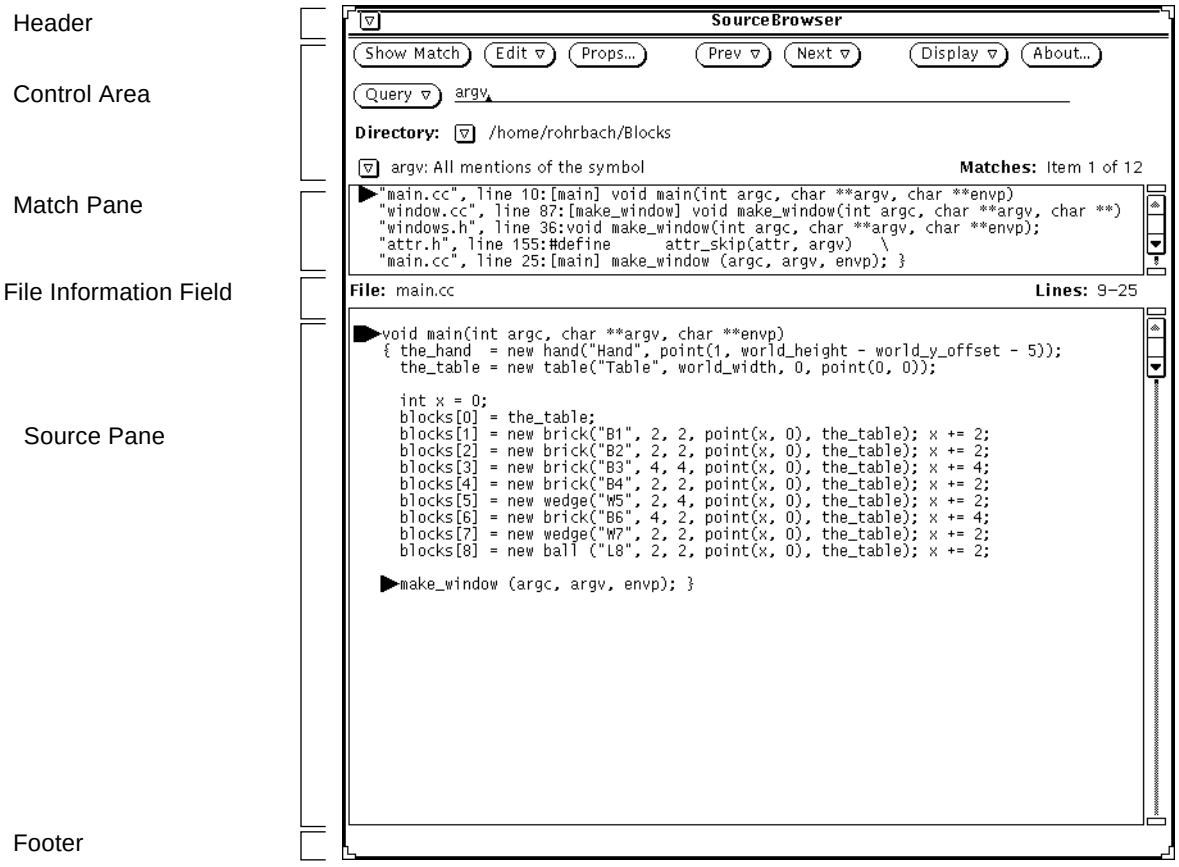
1.2 Graphical Overview

This section shows the icons, windows and menus of SourceBrowser. SourceBrowser consists of four main windows:

- SourceBrowser base window
- CallGrapher
- ClassGrapher
- ClassBrowser

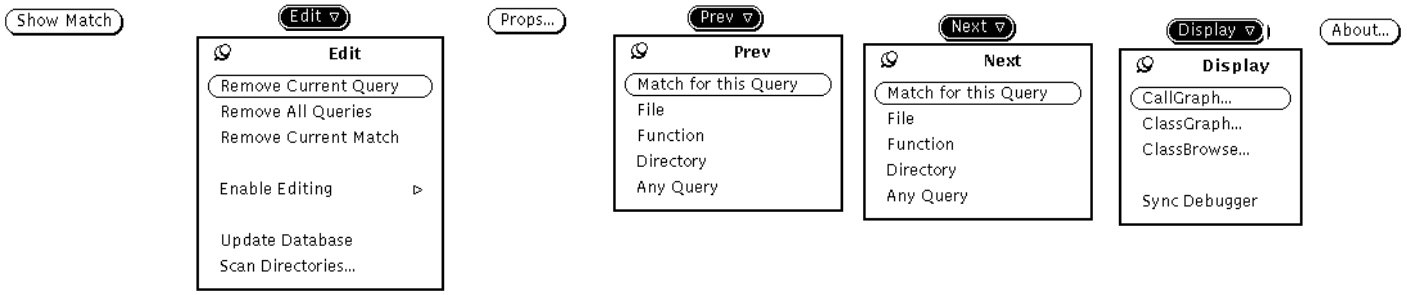
1.2.1 SourceBrowser Base Window

You issue queries and view matches in the SourceBrowser base window. You activate SourceBrowser from either the Manager or by entering **sbrowser** in a Command or Shell window.



SourceBrowser Buttons and Menus

SourceBrowser includes basic commands for issuing, deleting, and moving between queries.



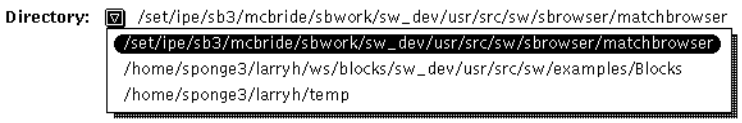
SourceBrowser also includes commands that allow you to limit and customize your search for symbols, to activate the CallGrapher, ClassGrapher, and ClassBrowser, and to obtain information about the SourceBrowser.

About Button

The About button displays a pop-up window that gives the version number and copyright information about the tool. Clicking SELECT (right mouse button) on the Comments button in this window displays another window in which you can write any comments you may have about the tool. The comments can be mailed back to Sun by clicking on Send.

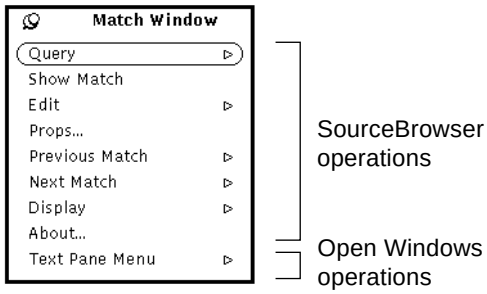
Directory History

SourceBrowser maintains a history of the directories that you have browsed in the directory menu.



Floating Menu

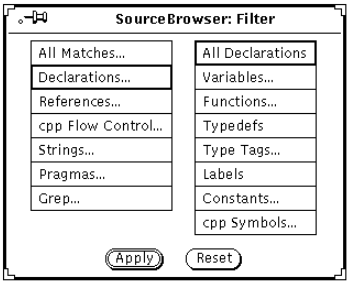
The SourceBrowser floating menu contains the same items as in the SourceBrowser button menus, plus an additional menu for performing standard OpenWindows text pane operations. You can display this menu from either the source pane or the match pane by pressing the SELECT button.



SourceBrowser Filter Window

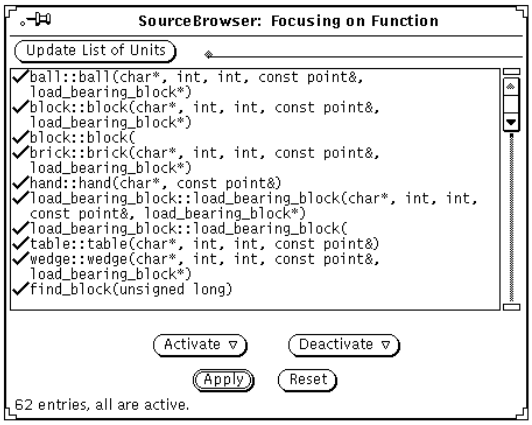
The Filter window lets you search for symbols based on how they are used in a program. You activate the Filter window by selecting Set Filter from the Filter submenu of the Query. (See Chapter 6 for details about Filtering.)

The settings in the Filter window are determined by the language(s) of the code you are browsing. The following C++ Filter window instructs SourceBrowser to search only for symbols that are used as declarations of variables. If you are browsing a mixed-language application, then the Filter window is a union of the settings available for the different languages.

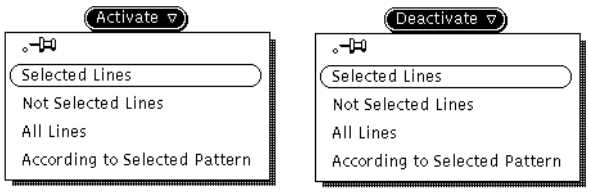


SourceBrowser Focus Window

The Focus window lets you restrict your query to instances of specific units of code, such as programs, functions, or libraries. You activate the Focus window by selecting from the Focus submenu of the Query menu. The following window is used to set the focus on specific functions, allowing you to restrict browsing to matches that occur in the functions you specify.

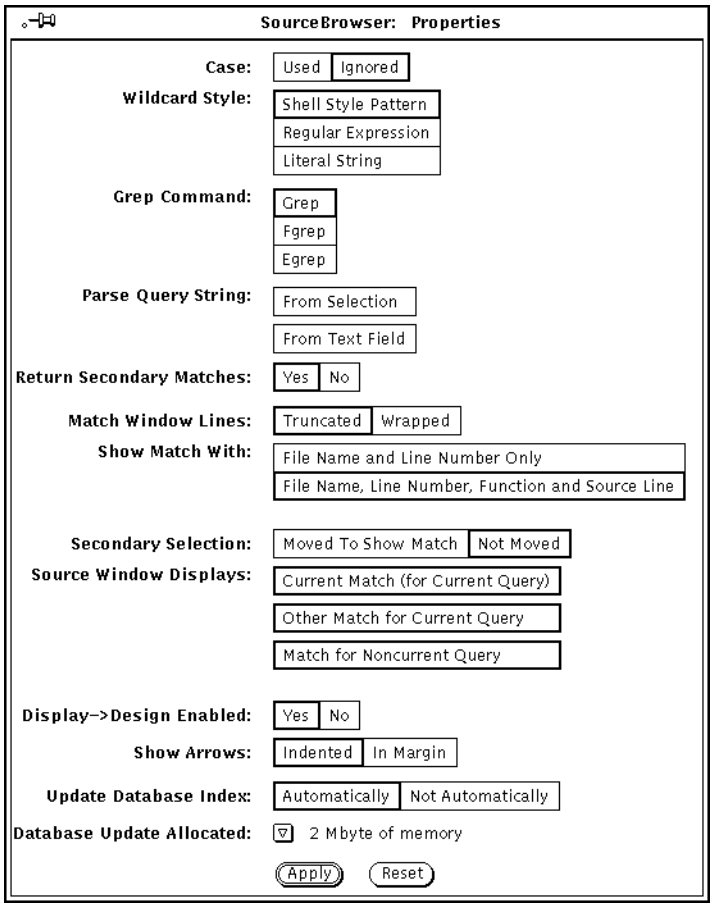


The Focus window has two menus for activating and deactivating the items you want to use in your query. (See Chapter 6 for details about Focusing.)



SourceBrowser Properties Window

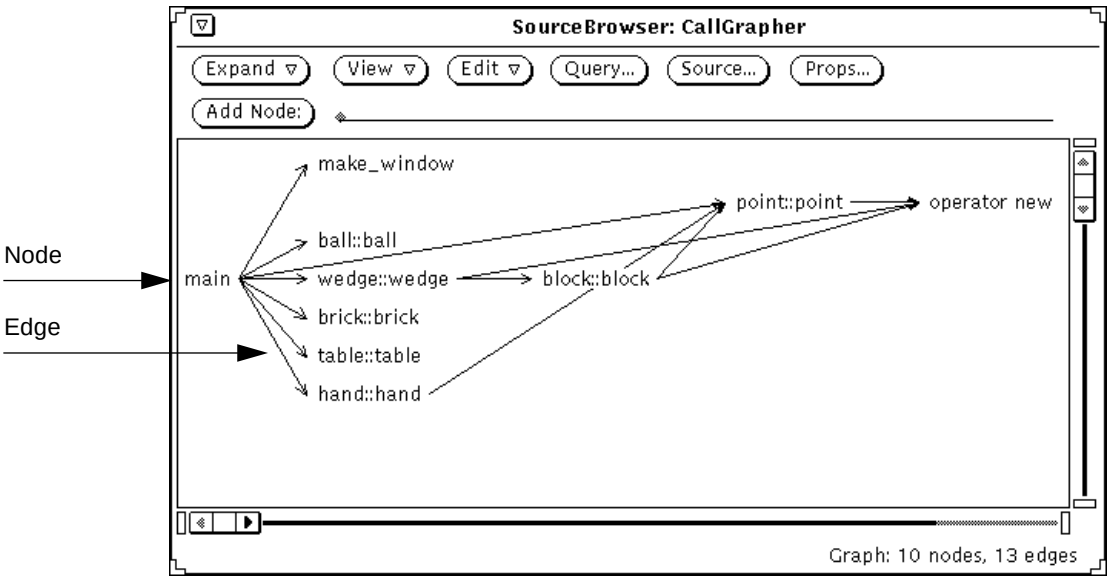
The Properties window lets you customize the behavior of SourceBrowser. You activate the Properties window by clicking on the Props button. The following Properties window shows the default values.



1.2.2 CallGrapher

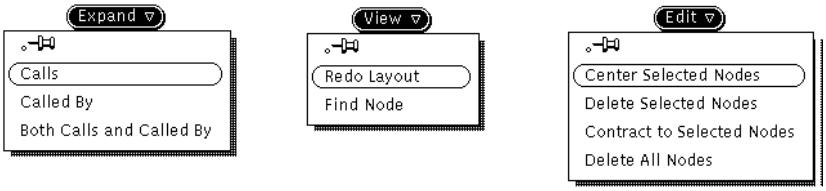
With the CallGrapher, you can graphically inspect the relationships of the functions in ANSI C, C++, FORTRAN, and Pascal programs. You activate the CallGrapher from SourceBrowser by choosing CallGraph from the Display menu.

The call graph consists of *nodes* and *edges*. A node is a name that represents a function. An edge is a line that connects two nodes. An edge represents a function call.

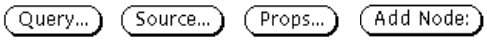


CallGrapher Buttons and Menus

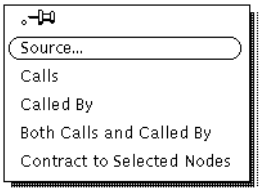
The CallGrapher menus allow you to expand, edit, and find a node in the graph.



The remaining buttons let you issue a query, show the source code where a node is defined, activate the Properties window, and add a node to the graph.

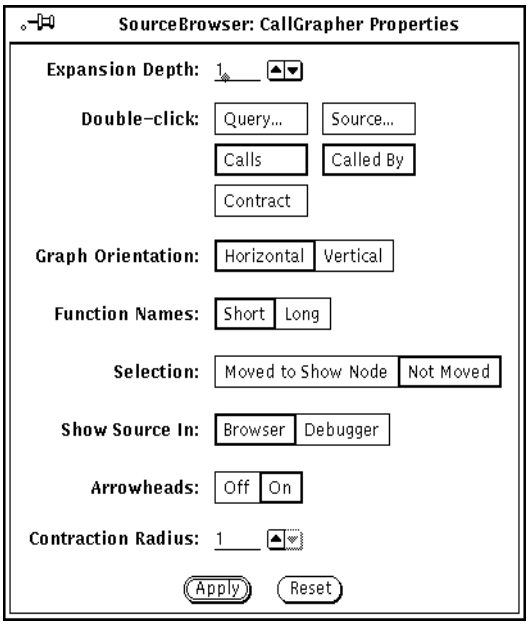


The CallGrapher floating menu contains some of the same items in the button menus. Press the right mouse button to display this menu in the graph pane.



CallGrapher Properties Window

The CallGrapher Properties window lets you customize the call graph. You activate the Properties window by clicking on the Props button. The following Properties window shows the factory-set default values.

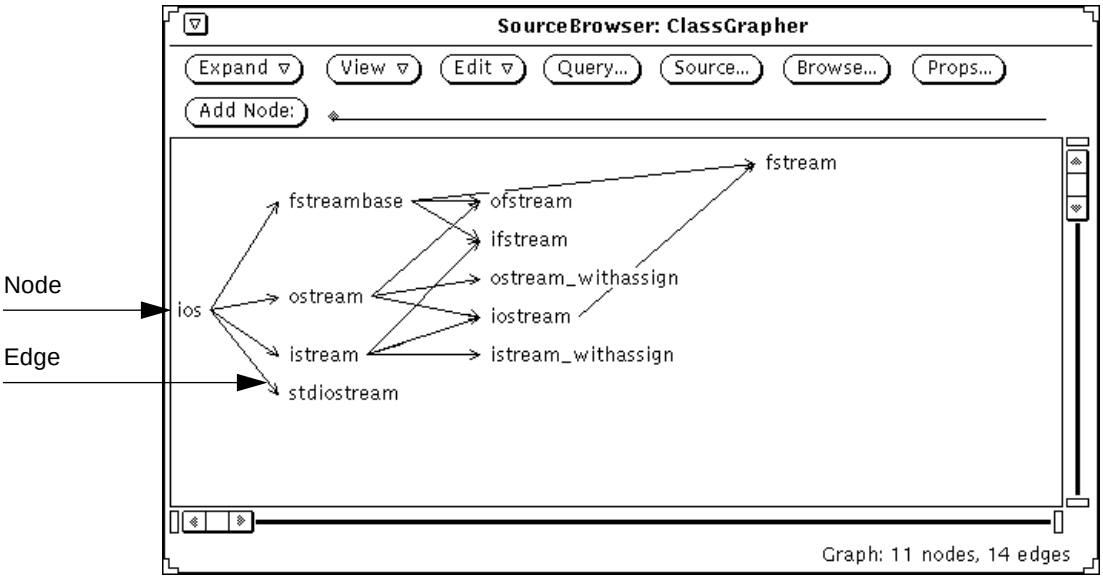


See Chapter 7 for details about CallGrapher Properties.

1.2.3 ClassGrapher

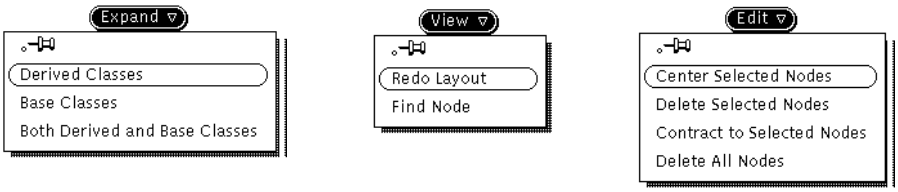
With the ClassGrapher, you can graphically inspect the inheritance structure of classes in C++ programs. You activate the ClassGrapher from SourceBrowser by choosing ClassGraph from the Display menu or by clicking on the Browse button in the ClassGrapher.

The class graph consists of *nodes* and *edges*. A node is a name that represents a class type. An edge is a line that connects two nodes; it represents class inheritance.

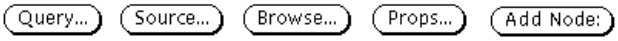


ClassGrapher Buttons and Menus

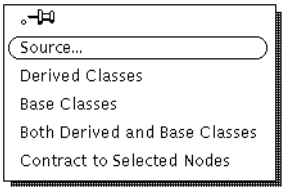
The ClassGrapher menus allow you to expand, edit, and find a node in the graph.



Other buttons let you query, show the source of, browse a class, activate the Properties window, and add a class to the graph.

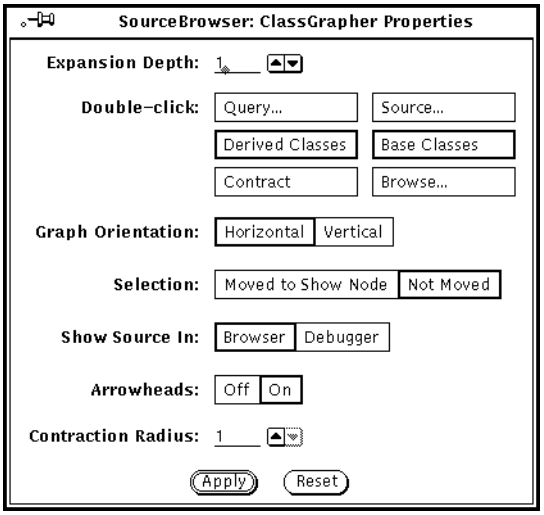


The ClassGrapher floating menu contains some of the same items in the button menus. You display this menu in the graph pane.



ClassGrapher Properties Window

The Properties window lets you customize the class graph. You activate the Properties window by clicking on the Props button. The following Properties window shows the factory-set default values.

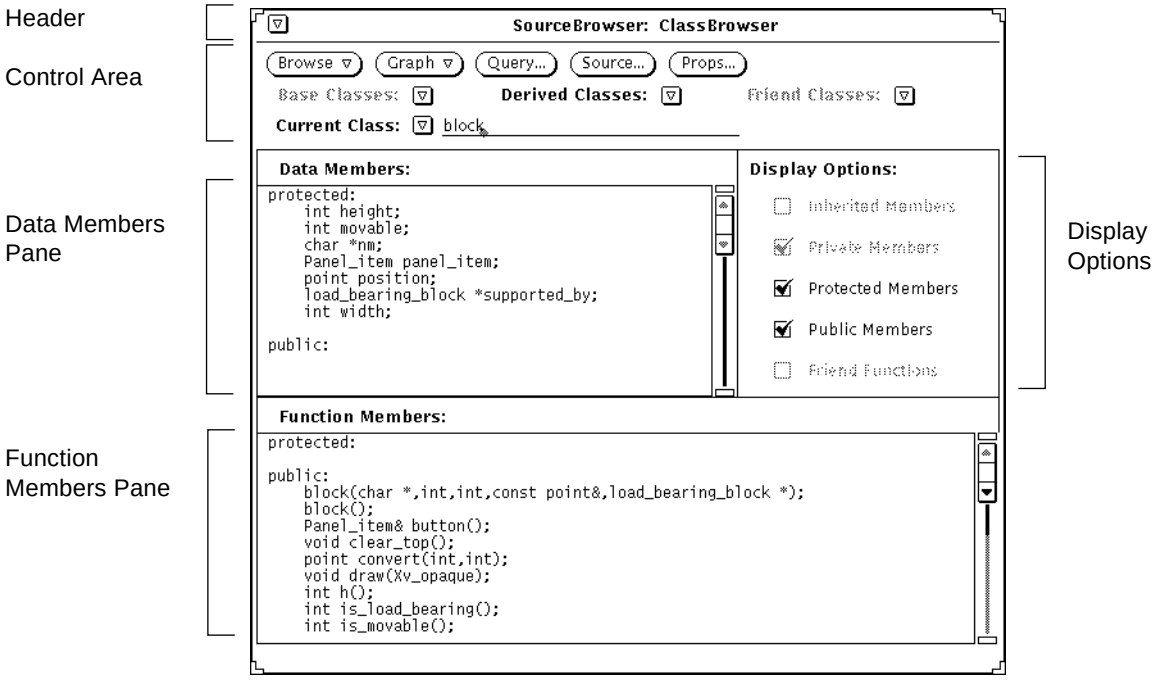


See Chapter 7 for details about ClassGrapher Properties.

1.2.4 ClassBrowser

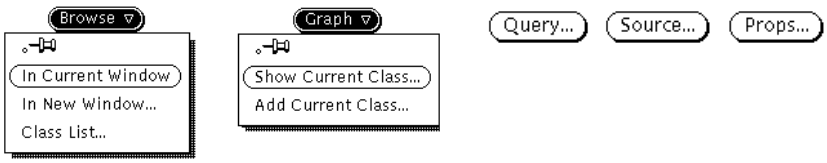
With the ClassBrowser, you can browse through C++ classes and their data and function members. You can also view the class interfaces and relationships.

You activate the ClassBrowser from SourceBrowser by choosing the ClassBrowse item from the Display menu.

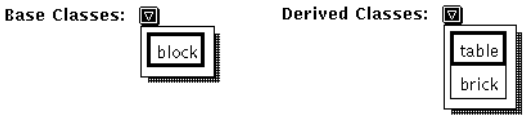


ClassBrowser Buttons and Menus

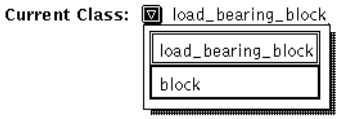
The ClassBrowser buttons and menus allow you browse, graph, query, or show the source of a class. You can also activate the Properties window.



The abbreviated menu buttons list the base, derived, and friend classes of the current class.

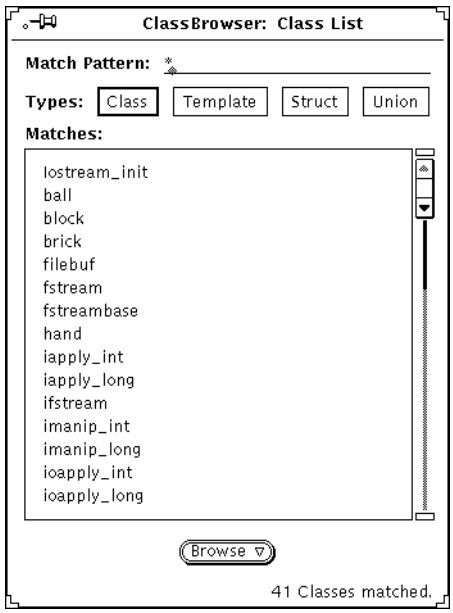


ClassBrowser maintains a history of the classes you have browsed in the Class menu.



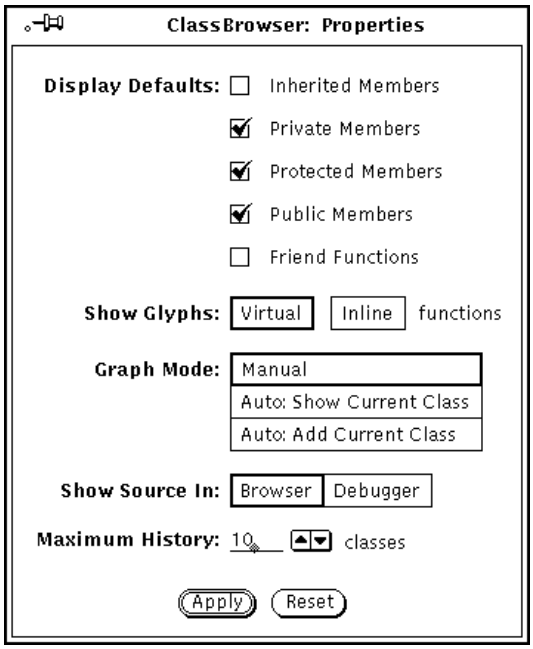
ClassBrowser List Window

The Class List window lets you list all classes, instantiated template classes, structures, and unions in your program. You activate the Class List window by choosing Class List from the Browse menu.



ClassBrowser Properties Window

The Properties window lets you customize the ClassBrowser. You activate the Properties window by clicking on the Props button. The following Properties window shows the factory-set default values.



See Chapter 7 for details about ClassBrowser Properties.

1.2.5 How the Browsers and Graphers Interrelate

Figure 1-1 shows how the SourceBrowser, CallGrapher, ClassGrapher, and ClassBrowser windows interrelate.

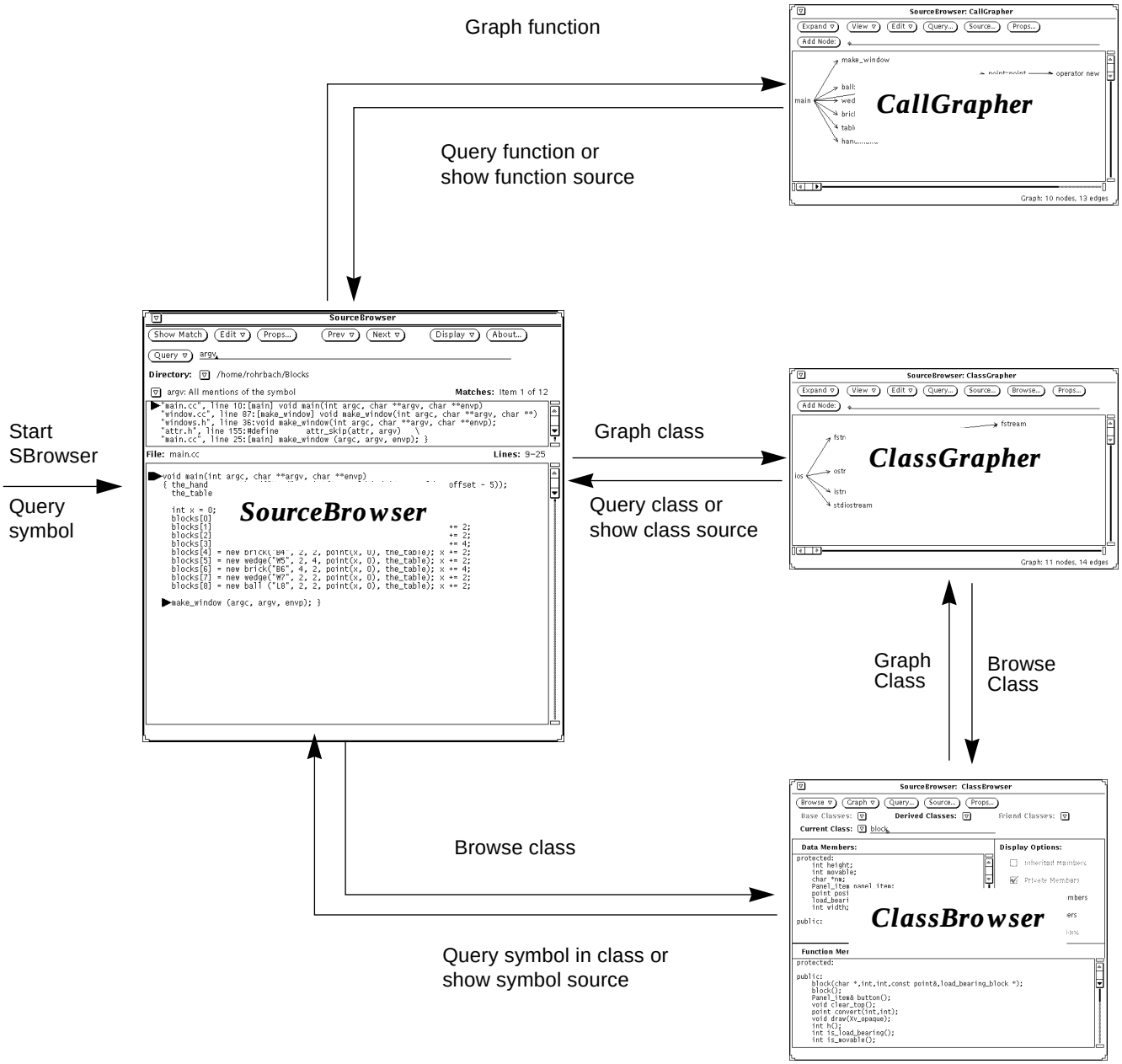


Figure 1-1 How SourceBrowser, the Graphers, and ClassBrowser Interrelate

1.3 Synopsis of SourceBrowser Tasks

When you start SourceBrowser, first decide the task you want to perform: query a symbol, graph a function, graph a class, or browse a class. Table 1-1 shows the steps involved in performing each task.

Table 1-1 SourceBrowser Tasks

Table 1:

Query Symbol	Graph Function/Class	Browse Class
1. Issue Query	1. Display Graph	1. Browse Class
Too many matches? Set a Filter or Focus, then reissue the query.	Graph function relationships or class hierarchy. Delete all unneeded nodes.	Show class list and data function members. View class interfaces and relationships.
2. Navigate Matches	2. Navigate Graph	2. Navigate Class
View matches with surrounding source code. Edit source code if necessary.	Traverse the nodes. Expand/contract graph to include nodes or reduce nodes.	Browse base, derived, or friend class of current class.
3. Remove Query?	3. Show Query/Source	3. Graph Class?
Remove query to free up SourceBrowser memory if done with query.	Query node or show node source in SourceBrowser base window.	Graph class hierarchy of class in ClassBrowser window.
4. Issue New Query?	4. Browse Class?	4. Show Query/Source
Return to step 1. Continue showing last query if not removed.	ClassGrapher only. Show base, derived, and friend classes in ClassBrowser.	Query class or show class source in SourceBrowser base window.

1.4 SourceBrowser Option Flags

`-editor`

The SourceBrowser uses ToolTalk™ for communication between and among components. When the SourceBrowser is started with the `-editor` flag, it does not present a source pane. Instead the communication ToolTalk messages can be used by any ToolTalk-aware editor to provide an alternative displayer of source code. The ToolTalk protocol is described in the *Integrating Applications with the SPARCworks 3.0 Toolset* document and is supported by SunSoft.

Currently SunSoft does not provide an editor that supports ToolTalk. However, externally supplied editors are available. The Xemacs™ editor and Eos™ packages are used in the descriptions in Chapter 8.

`-query` <*symbol*>

When using the query symbol flag option, the SourceBrowser peforms a `query` on <*symbol*> immediately after startup.

Part 2—Basic Browsing

SourceBrowser Basics 2

This chapter describes how to create the database required by SourceBrowser. It also includes the procedures for starting SourceBrowser in a window environment. See Appendix B, "Issuing a Command-line Query," if you want to use SourceBrowser in the command-line environment.

This chapter is organized into the following sections:

Generating a SourceBrowser Database	*page 25*
Setting up Sourcebrowser for Large Applications	*page 32*
Starting SourceBrowser	*page 34*
Changing Directory	*page 36*
Getting Help	*page 40*
Quitting SourceBrowser	*page 39*

2.1 Generating a SourceBrowser Database

The SourceBrowser obtains the information it uses for its displays from a database of information that describes the static structure of your application. It has three different levels of operation depending upon the way in which the database was generated:

1. Compiler-generated Database

If the database was generated by compiling your source code with the SourceBrowser compilation flag, **-xsb** for C, or **-sb** for C++, Fortran, or Pascal, then all features of the SourceBrowser are operational.

The information that is included depends on the particular compiler. In general, only identifiers and strings are included; reserved words are not included.

If you cannot or do not choose to compile the source code, an option exists to create a less complete, but still useful, database. This option is called scanning and is performed via the `sbtags` command. It is discussed in detail in this Chapter under, *Collecting Information without Compilation.*

2. *`sbtags`* Generated Database

If the database was generated by the `sbtags` program, you can perform queries on functions and global variables and can display function calls. In this case, SourceBrowser does not have any information for browsing or graphing C++ classes.

3. No Database

When you have no database, `grep` queries are still available. In this case, the SourceBrowser can be used as a convenient and easy-to-use graphical user interface for `grep`. Queries using `grep` are discussed later in this Chapter under, *Query/Grep Command*

2.1.1 Comparison of Database Types

The following summary lists the advantages / disadvantages of the three levels of operation:

Compiler Generated Database

- All SourceBrowser features are available.
- The database size is usually greater than the size of your source code.
- The database contents are semantically correct because they are generated by the compiler.

`sbtags` *Generated Database*

- ClassBrowser and ClassGraph features are not available.
- The database does not contain information on all symbols and strings. It contains information on functions, classes, types, and global variable definitions, and calls to functions.
- The `sbtags` program runs faster than the compiler.
- The database size is much smaller than the size of your source code.
- The database contents is not guaranteed to be semantically correct because the `sbtags` program performs only simple syntactic and semantic analysis, and may sometimes be in error.
- The `sbtags` program always generates a database even if the source code it cannot be compiled because it is incomplete or semantically incorrect.

No Database

- CallGraph, ClassGraph, and ClassBrowser features are not available.
- Database-based queries are not available, but `grep` queries are.

Mixed database

Even though the SourceBrowser provides different levels of functionality, according to the method by which the database is generated, it does not require that a single database be uniform in content. You can use the SourceBrowser when some files have been compiled, others have been scanned by `sbtags`, and others have not been processed at all. The results of your SourceBrowser queries, and other operations, will depend upon the method of database generation for each file. Because of the availability of the mixed-mode of operation, it is easy to start using the SourceBrowser on an existing application even if it is very large.

- Even without compiling or scanning your source code, you can use the SourceBrowser as an easy-to-use graphical user interface to `grep`.
- To quickly expand the capabilities of the SourceBrowser, run `sbtags` on your source code.

- When you have new, un-compilable source files, run `sbtags` on them to add them to to the database.As each file of your application becomes compilable, and is compiled, full semantic information describing it will be added to your database.

As each file of your application becomes compilable, and is compiled, full semantic information describing it will be added to your database.

2.1.2 *Using* `make` *to Turn on SourceBrowser*

♦ **Add the SourceBrowser option to the language macro in your Makefile.**

Table 2-1 Makefile Macros

If Your Program is Written in	Add this Option	To this Macro
ANSI C	`-xsb`	`CFLAGS`
FORTRAN 77	`-sb`	`FFLAGS`
Pascal	`-sb`	`PFLAGS`
C++	`-sb`	`CCFLAGS`
Assembler	`-b`	`ASFLAGS`

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 or updates an index for this database and 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 is performed faster.

To automatically rebuild the database index every time you run `make`:

♦ **Add the following lines to your makefile:**

```
.DONE: query
query:
        -sbcleanup
```

`sbcleanup` marks for deletion, any obsolete SourceBrowser files in the current database.

2.1.3 *Not Using* `make` *to Turn on SourceBrowser*

To turn on SourceBrowser if you do not use `make`:

♦ **Add the SourceBrowser option to the compiler command line.**

Table 2-2 Compiler Command Lines

If Your Program is Written in	Use this Command
ANSI C	**`acc -xsb`** *filename*`.c` *<other options>*
FORTRAN 77	**`f77 -sb`** *filename*`.f` *<other options>*
Pascal	**`pc -sb`** *filename*`.p` *<other options>*
C++	**`CC -sb`** *filename*`.cc` *<other options>*
Assembler	**`as -b`** *filename*`.S` *<other options>*

2.1.4 *Scanning: Collecting Information without Compilation*

The `sbtags` command provides a quick and convenient method, called scanning, for collecting browsing information from source files and enables you to collect browsing information about programs that do not compile completely. The `sbtags` command collects a subset of the information available through compilation. The reduced information restricts some browsing functionality.

The follow restrictions currently exist:

- Cannot issue queries about local variables.
- Cannot browse classes.
- Limited ability to issue complex queries.
- Limited ability to focus queries.

- Scanned database information may be less reliable than compiled information.

Once a file has been scanned, it often need not be scanned again because the database can adjust to small changes.

Scanning is based on a lexical analysis of the source file. This will not always correctly identify all the language constructs but it will operate on incompilable files, and it is faster than having to recompile.

Scanning can be invoked automatically (via the auto-scan feature, invoked whenever sbrowser is started in a directory without a database). Scanning can also be invoked manually using the `Edit->Scan Directories` menu option or by using the `sbtags` <*list-of-files*> command.

The scanner recognizes global definitions for variables, types and functions. It also collects information on function calls. No other information is collected (in particular, other semantic information for complex queries is not collected).

The scanner works for C, C++, Pascal, and Fortran. C++ member function calls are recognized only when invoked explicitly as `ClassType::MemberFunctionName()`; better handling requires semantic analysis of the program and can be accomplished using the compiler with the **-sb** flag.

Users of **ctags** and **etags** will recognize that the functionality of `sbtags` is similar to these programs (except for the CallGrapher and ClassGrapher information). A typical user of the scanner will intermix direct queries to the database, for definitions and graphing, with queries using the `grep` filter for more detailed investigation. Note that the scanner does not collect any information for the class browser.

- Run simple queries for symbols.
- Run limited filtered and focused queries.

To generate the SourceBrowser database using `sbtags`:

♦ **Type** `sbtags`**, followed by the name of the file for which you want to generate the database.**

For example, the following command generates a SourceBrowser database for the ANSI C file `hello.c`.

```
venus% sbtags hello.c
```

This command generates the database for all files in the directory.

```
venus% sbtags *
```

For more information on `sbtags`, see the `sbtags`(1) man page.

2.1.5 Query/Grep Command

When you select the Query / Grep option, the SourceBrowser executes the `grep` program and displays the matches found by the `grep` command in the Match panel. All of the standard SourceBrowser features for browsing a query apply to a `grep` query. The source file containing current match is displayed in the **Source** window. You can view different matches by using the **Prev** and **Next** menus. `Grep` queries are saved together with other queries, and you can go back to any query by using the **Query History** abbreviated menu button.

The SourceBrowser can interface with any of the standard grep programs, `grep, egrep, or fgrep`. You can select whichever you prefer by specifying it in the SourceBrowser's **Property** sheet. When you select the **Query/Grep** command, the SourceBrowser executes the `grep` command found through your **$PATH** environment variable. Consequently, if you use a private version of `grep`, rather than the standard system version of `grep`, you must insure that it will be found via **$PATH**.

The grep command line includes:

- A variety of optional command flags.
- A regular expression or string to search for.
- A list of files to search in.

(See the man page for the grep program that you use.)

1. To use the SourceBrowser's **Query/Grep** command:

- **Type in a `grep` command line in the Query text entry field.**
  No `grep` flags can be used because they change the format of the output.
- The list of files is optional. If no list is specified, the set of files referenced by the SourceBrowser database is searched.

2. Select the Grep option from the Query menu.

Alternatively, instead of typing the grep command line in the query text entry field, you may use the mouse to select it from any window on your desktop.

If you plan to do a series of grep commands, you can simplify the process for using grep by either of two methods:

1. Pin the Query menu on the desktop so that you can easily SELECT the Grep option.

2. Use the standard OpenLook technique for changing the default menu option: move the cursor over the Query menu, hold down the MENU button, then move the cursor over the Grep option. CLICK on the Control key and release the button. The default Query menu option is now Grep. Subsequently, you can invoke the Grep option by simply clicking SELECT on the Query menu.

2.2 Setting up SourceBrowser for Large Applications

The SourceBrowser uses data that has been stored in a database by either the compiler or the `sbtags` program. By default, when you compile or scan (`sbtags`) your source code, the data base is created in the same directory in which the compiler is run. If all the source files for your application are kept in a single directory, the default behavior usually works correctly. That is - you usually execute the compiler in that same directory, and run the SourceBrowser in that same directory.

Note: It is not necessary for header files that have been included by the `cpp #include` macro to be in the same directory. For example, standard system header files are located in `/usr/include`.

Note: In this discussion, "source files" refers to those files which are specified as input to the compiler. These files are typically identified by the suffix `.c` for C, `.C` or `.cc` for C++, `.f` for Fortran, or `.p` for Pascal. `#included` header files are typically identified by the `.h` suffix.

If you have a moderate-to-large size application, you are likely to keep your source files in several different directories, and you are likely to execute the compiler in each of these directories.

As a result, the default compiler behavior will generate a separate SourceBrowser database in each of these directories. In its default behavior, the SourceBrowser looks at only one database at a time. Consequently, when you perform a query, or other operation, you will only see data for that part of your application that is located in the current directory. There are two methods that can be used to override this default behavior:

1. Merged database: the most space- and time-efficient way to set up your database is to merge all of the separate databases into one. Perform the following steps to achieve the merge:

Identify, or create, a directory where you want the single database to exist. (Note: The database size will be equal to, or greater than, the size of your source code files.) Call this <`database_directory`>.

In each directory in which the compiler executed (usually all directories that contain source files), create a symbolic link to the database:

```
% cd <directory in which the compiler is executed>
% ln -s <database_directory>/.sb
```

2. Imported Databases: You can use the `.sbinit` file to tell the SourceBrowser to read more than one database. (As noted in Method 1, above, a Merged Database is the most space- and time-efficient. However, if your application is very large, you might be required to set many symbolic links. If that task becomes too cumbersome, you can use an Imported Database instead.) To do this:

Identify the place to put your `.sbinit` file (usually your home directory).

For each directory in which the compiler is executed, add an `import` command to the `.sbinit` file.

`import` <*absolute path to the directory in which the compiler executed*>

Set your SUNPRO_SB_INITFILE_NAME Environment variable to point to your `.sbinit` file. For example, you would add the following line to your `.login` file:

`setenv SUNPRO_SB_INIT_FILE _NAME` <*path to .sbinit file location*>`/.sbinit`

2.3 Starting SourceBrowser

You can start SourceBrowser from one of the following:

- Manager
- Command or Shell tool window

Since SourceBrowser is an integrated component of the toolset, try working with the Manager. If you use the Manager, then you can use SourceBrowser in conjunction with the Debugger. Specifically, you can use SourceBrowser to search for a variable; then, set a Debugger breakpoint at the appropriate occurrence of that variable. You can also select a symbol in the CallGrapher, ClassGrapher, or ClassBrowser window, and show its source in the Debugger window.

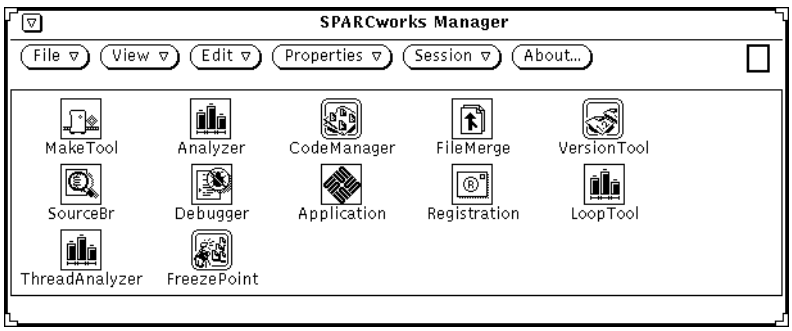
2.3.1 From the Manager

To activate the Manager:

♦ **Type `sparcworks` or `proworks` at the prompt in a Command or Shell tool.**

To activate SourceBrowser from the Manager:

♦ **Double-click on the SBrowser icon in the Manager window, or drag and drop the icon onto the workspace.**



For details on using the Manager with SourceBrowser and other tools, see the *Managing the Toolset* manual.

2.3.2 From a Command or Shell Tool Window

To activate SourceBrowser from a Command or Shell tool window:

♦ **Type** `sbrowser&` **at the prompt in the Command or Shell tool window.**

When you start SourceBrowser from the command line, you can specify any of the standard OpenWindows `-scale` options (`small`, `medium`, `large`, or `extra_large`). The default is `-scale medium`.

2.3.3 SourceBrowser Base Window

The following figure shows the base window similar to that displayed when you first invoke SourceBrowser.

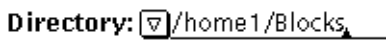
2.4 Changing the Directory

By default, SourceBrowser browses all source files described by the database in the current working directory. The SourceBrowser can be set up to browse files in different directories. The path to the current working directory is displayed in the Directory field above the match pane.

If you start SourceBrowser in your home directory, then you need to change to the directory in which you compiled your program.

To change the current working directory from within SourceBrowser:

1. **Click on the name of the current working directory.**
   SourceBrowser underlines the directory name to indicate that the field is editable.



2. **Enter the new path name in the text field and press Return.**
   You can enter a relative or absolute path name. SourceBrowser always displays the absolute path name. When you press Return, SourceBrowser removes the underline.

3. **When you change directory, SourceBrowser clears:**

- All queries in the SourceBrowser base window
- All graphs in the CallGrapher and ClassGrapher
- All classes in the ClassBrowser

If you have issued a query, graphed a function or class, or browsed a class in the current working directory, SourceBrowser pops up a notice asking if you really want to change directory.

The Directory menu contains a history of all the directories you have used. You can use this history menu to quickly move between directories. The history is maintained until you quit SourceBrowser.

To return to a previous directory:

- **Choose the directory name from the directory history menu by clicking MENU on the Down arrow symbol.**

.

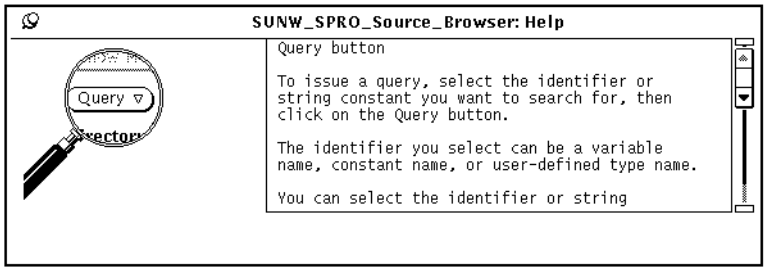
2.5 Getting Help

The SourceBrowser Help window provides on-line help for each control, window, and pane displayed on the screen.

To get help on using SourceBrowser:

- **Move the pointer onto the object for which you want help and press the the Help key or the F1 key.**

In this example, the user asked for help on the Query button.



SourceBrowser relies on finding its help files, `SourceBrowser.info`, `CallGraph.info`, `ClassGraph.info`, and `ClassBrowser.info`, in the `lib/help` directory next to the directory that contains `sbrowser`. If you install in a nonstandard way so that the files are not found where expected, then the system returns an error message when you ask for help.

To instruct SourceBrowser to search for the help files in another directory:

- **Set the HELPPATH environment variable to the desired directory.**

In this example, the user set **HELPPATH** to the standard OpenWindows locations.

```
venus% setenv HELPPATH /set/pubs/ow3/lib/locale:
/set/pubs/ow3/lib/help
```

2.6 Quitting SourceBrowser

To quit the SourceBrowser base window:

♦ **Choose Quit from the SourceBrowser frame header menu.**

When you quit the SourceBrowser base window, only that window closes. Any Grapher and ClassBrowser windows that are open *remain* open and fully functional. You can easily reopen the SourceBrowser base window by clicking on the Source or Query buttons in the Graphers or ClassBrowser.

If you pin a menu from the SourceBrowser base window, and then delete the window, the menu remains open if either the Graphers or ClassBrowser are running. When you choose an item from the menu, the base window reopens.

To quit the SourceBrowser application, you must quit all open Browser and Grapher windows that belong to the SourceBrowser base window. See Chapter 5, **"**Graphing Functions and Classes,**"** for information on the Graphers and Chapter 6, **"**Browsing Classes,**"** for more information on ClassBrowser.

Issuing A Simple Query 3

This chapter describes how to issue a simple query. A query is an instruction to SourceBrowser to find the uses of a specified symbol.

This chapter describes how to search for identifiers, the most common SourceBrowser operation. See Chapter 9, **"**Issuing More Advanced Queries,**"** for instructions on searching for string constants and using wildcards in queries.

This chapter is built around a C++ program called *Blocks*. See Chapter 1, **"**Introduction to SourceBrowser,**"** for more information on the *Blocks* program .

This chapter is organized into the following sections:

Issuing the First Query	*page 41*
Issuing Subsequent Queries	*page 46*
Tips on Issuing Queries	*page 47*
If Your Query Fails	*page 49*

3.1 Issuing the First Query

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

1. **Select, or type in, the identifier that you want to search for.**
   The identifier can be any name used in the program. For example, a variable, function, type, constant, or macro. You can type the identifier in the Query Text field or select an identifier in the SourceBrowser window or in another window.

2. **Select a query from the Query menu.**

Four standard types of query are listed:

- Declarations — Search for lines where the specified symbol is defined or declared
- References — Search for all lines that reference the specified symbol
- Grep — Search the source files for all occurances of the specified text string
- All — All of the above

In general, you cannot query for reserved words. The exception is language-defined type names in ANSI C and C++. For example, you can query on `int`, `float`, `double`, or `long` in ANSI C or C++ programs; you cannot query on `integer` or `print` in FORTRAN programs. See Table 3-1.

Table 3-1 What You Can Query For

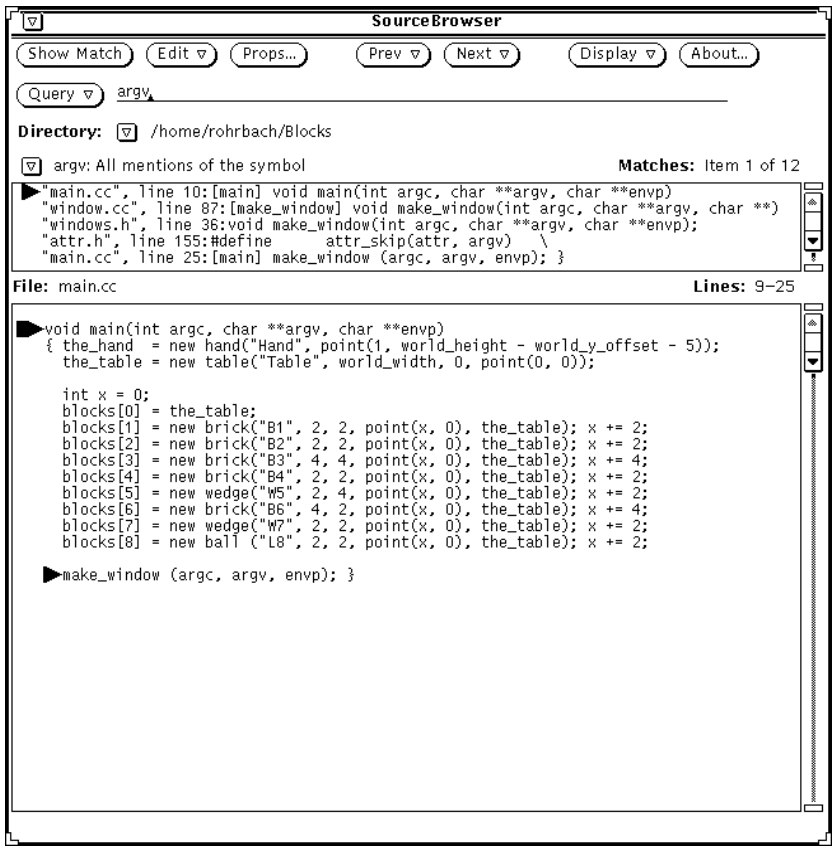
Collector	Identifier (variable, constant, or user-defined type name)	Reserved Words	String Constants
`cc` and `CC`	yes	Language-defined type names	yes
`f77`	yes	no	yes
`pc`	yes	no	yes
`as`	yes	no	yes
`sbtags`	yes	no	yes

If you are unsure of what to query for, then begin by selecting an identifier central to your program. For example, if your program is written in ANSI C or C++, then you might query for `main`. For other languages, choose an identifier used early in the program.

See “Querying for String Constants” in Chapter 9, for information on searching for strings.

The symbol for which SourceBrowser is actively searching is called the *current query.* All symbols in the code that are identical to the query are called *matches.* The first occurrence of the symbol SourceBrowser finds is called the *current match.*

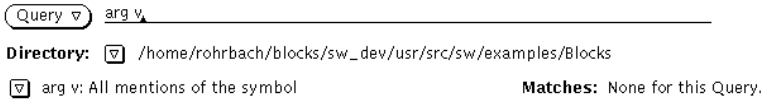
In this example, the user issued a query on the symbol `argv`. SourceBrowser returned 12 matches.



The following sections describe the contents of the SourceBrowser window after you issue a query.

3.1.1 Query Line

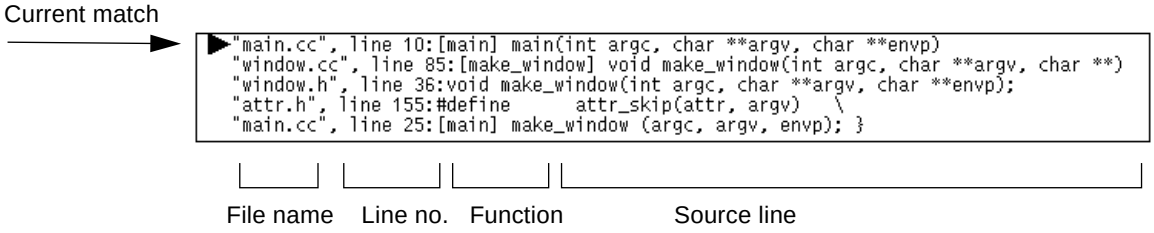
After you issue your query, the Query Text field may contain additional information on your query. For example, if you query for a symbol with a white space in it, then SourceBrowser displays the symbol name. Or, if you turn off Case in the Properties window, then SourceBrowser includes `[No case]` in the Query Text Field.



See “Issuing Case-Insensitive Queries” on page 159 for more information on the SourceBrowser Case property.

3.1.2 Match Pane

The match pane displays the file name, line number, function name, and source line of all matches found by the current query. The current match is marked with a large, black arrow.



In the match pane, SourceBrowser sorts the matches as follows: definitions first, declarations last, and everything else in between. Within these three groups (definitions, declarations, and other), SourceBrowser sorts the matches alphabetically by file name and then by line number.

Occasionally, SourceBrowser displays duplicate matches for the same symbol in a header file. The reason for this duplication is twofold:

- A particular header file may have been included in more than one source file.
- The context in which the header file is included in one source file may be different from the context in which it is included with another source file.

3.1.3 File Information Field

The file information displays the following:

- The name of the source file displayed in the source pane
- The line numbers of the first and last lines of the source code in the source pane

3.1.4 Source Pane

The source pane displays the portion of source code that contains the current match. The source pane contains arrows that identify matches found by the current and other active queries.


The current match is marked with a large black arrow in the source pane margin.


All other matches found by the current query are marked with a smaller arrow.

The Source Pane is not present when the SourceBrowser is started by using the `-editor` flag. In that case, the SourceBrowser generates ToolTalk (TM) messages that could be used by an external editor to present the source code.

3.1.5 Footer

The footer displays messages that provide you with information about the query or tell you about an error. To get more information on the message, press the Help key.

3.2 Issuing Subsequent Queries

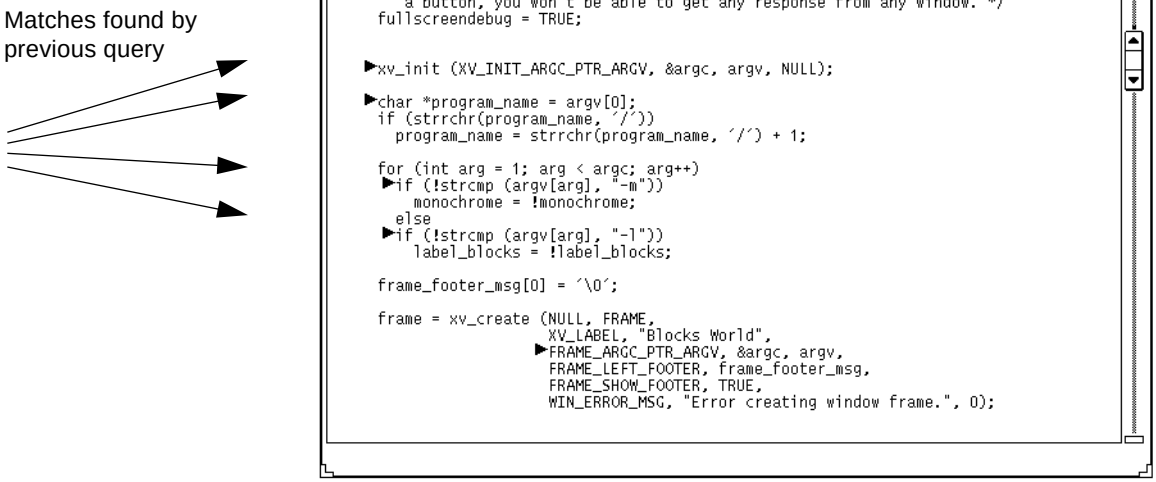
Issuing subsequent queries is similar to issuing the initial query. The only difference is that you now have the option of choosing as your search symbol text that is displayed in the match pane or source pane.

To issue a subsequent query:

- **Select the identifier you want to search for, then click on the Query button.**

You can also type the identifier in the Query Text field, then press Return.

In this example, the user selected `make_window`, then clicked on the Query button. SourceBrowser found three instances of `make_window`. The small arrows next to `argv` indicate matches found by the previous query for `argv`.



If you do not want to see the arrows for previous queries, then you can instruct SourceBrowser not to show them in the source pane. You do this by turning off Match for Non-Current Query in the Properties window. The glyphs are immediately removed from the source pane.

3.3 *Tips on Issuing Queries*

The following is important additional information about issuing a query.

- When using the Query Text field to issue a query, you can simply type the query in the Query Text field and press Return rather than clicking on the Query button.
- If the SourceBrowser Secondary Selection property is set to Moved to Show Match, then SourceBrowser underlines the current match in the source pane. Underlinig can be useful when there is more than one match per source line.
- In C++ programs, SourceBrowser performs queries for operator functions on the operator symbol. For example, to query on the function operator "+", you must type the character, "+".

  To query on the function "*":

  Set **Wildcard Style** to **Literal** in the **Properties** window

  Type ***** in the **Text** field and press Return.

  Otherwise, SourceBrowser treats the "*" as a shell-style pattern and matches every string and symbol in the database. Such a query takes a long time and may exhaust all available swap space, causing SourceBrowser to abort. Once you start the query, you will not be able to exit SourceBrowser.

  See "Using Wildcards in Queries" on page 154 for more information on pattern recognition in SourceBrowser.
- When there is more than one match per source line, SourceBrowser indicates this situation in the footer. SourceBrowser only displays one arrow per line with match(es).
- You can halve the time it takes to respond to a query by not including the source code in the match pane. You do this by setting Show Matches With to File Name and Line Number Only in the Properties window. See Chapter 10, "Customizing SourceBrowser,"for details on the Properties window.
- To limit the number of matches you receive, you can use the Filter panel to search for items based on how they are used in a program. You can also use the Focus window to limit the search to a subset of your code.. See Chapter 6, "Restricting the Match Set," for details.

3.4 If Your Query Fails

The SourceBrowser footer displays messages that provide extra information or tell you about an error. If your query fails, then check the following list:

- If SourceBrowser returns the message `No Matches`, then it cannot find any occurrences of the symbol you specified. The Show Match, Prev, and Next buttons are inactive and are grayed out. Check that you have typed the symbol correctly in the Text field. Click on the Edit button to remove the query, then issue the query again.
- Check to see that you have the correct database.
- Check to see that the files were compiled with SourceBrowser options.
- If SourceBrowser returns the message `Attempt to query for the empty string`, then you have not specified a symbol to search for. Repeat steps one andtwo in the section**, "**Issuing the First Query,**"** in this chapter.
- If SourceBrowser indicates there is no database in the directory, then you need to change to the directory in which you compiled your files. See Chapter2, **"**Changing Directory,**"** for instructions.
- 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), then remove the `.sb` directory (that is, the directory containing the SourceBrowser database) and recompile everything with the SourceBrowser option. If you receive a message stating that the database is locked, then you can either issue the `sbquery` command with the `- break_lock` option or remove the `.sb` subdirectory and recompile. See Appendix B, **"**Issuing a Command-line Query",**"** for a description of the `-break_lock` option.
- If you receive a message stating `Request for xxx bytes of memory failed`, then you have run out of swap space.

  To create more swap space or to free up swap space, use the `mkfile`(1M) and `swap`(1M) commands.

  To determine which processes occupy significant swap space, use the `ps`(1B) command.

  To determine how much swap space you have, use the `sar`(1M) command.

Viewing Matches 4

Once you issue a successful query, you will want to view the matches in the source code. A match is a symbol in the code identical to the query.

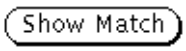
This chapter is organized into the following sections:

Redisplaying the Current Match	*page 51*
Viewing the Next Match	*page 52*
Viewing the Previous Match	*page 54*
Viewing a Selected Match	*page 56*
Moving to a Match from Any Query	*page 56*
Moving Between Queries	*page 59*
Removing Matches	*page 59*
Removing Queries	*page 60*

4.1 Redisplaying the Current Match

Often, when using SourceBrowser, you will scroll through either the match or source pane so that the current match is no longer displayed.

To redisplay the current match in the match and source panes:



♦ **Click on the Show Match button.**

4.2 Viewing the Next Match

The *next match* is the match that appears in the source code after the current match.

To view the next match:

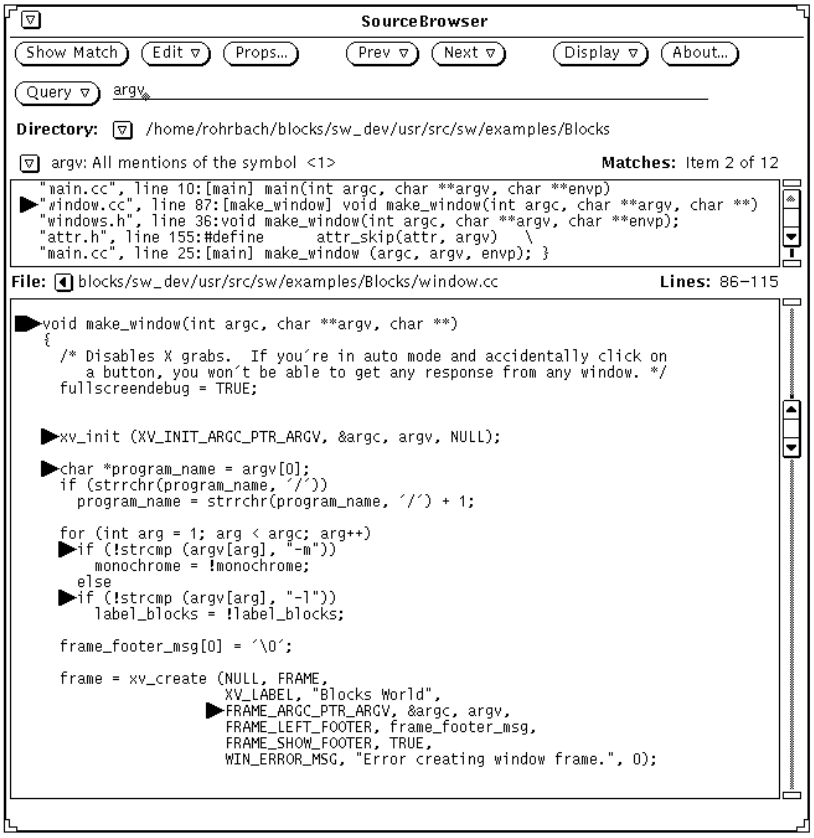
Next ▽

- **Click on the Next button.**
  SourceBrowser makes the next match in the match pane the current match and displays its surrounding source code in the source pane.

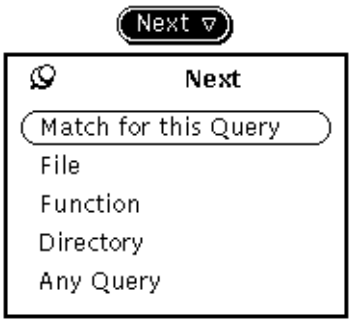
In the following example, the user first issued a query on the symbol `argv`. The user then instructed SourceBrowser to display the next match of `argv`, which it found in the `window.cc` file.

Next match of argv is second in the list of matches

Current match is in the file window.cc



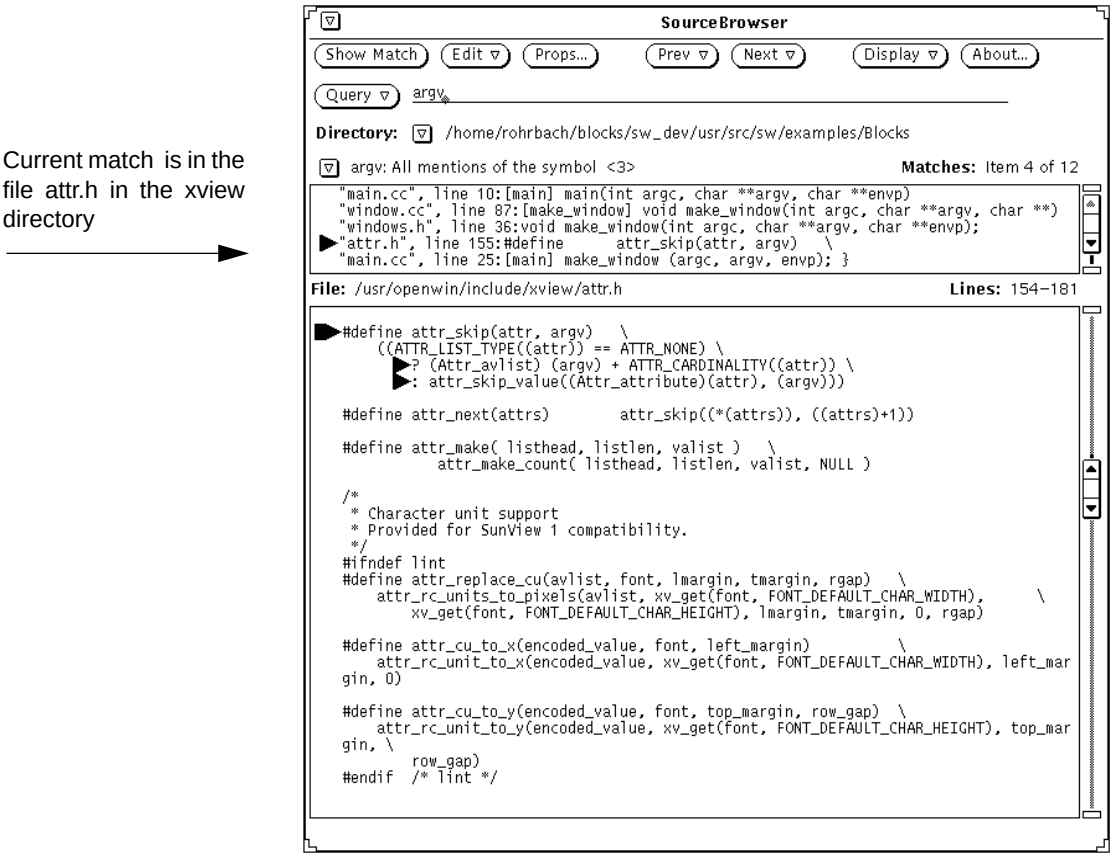
To display the first match that occurs in the next function, file, or directory::



- **Choose File, Function, or Directory from the Next menu.**

In this example, the user chose Directory from the Next menu. SourceBrowser moved the current match to the first occurrence of argv in the `attr.h` file. This file is in the `xview` directory.

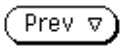
Current match is in the file attr.h in the xview directory



4.3 Viewing the Previous Match

The *previous match* is the match that appears in the source code before the current match.

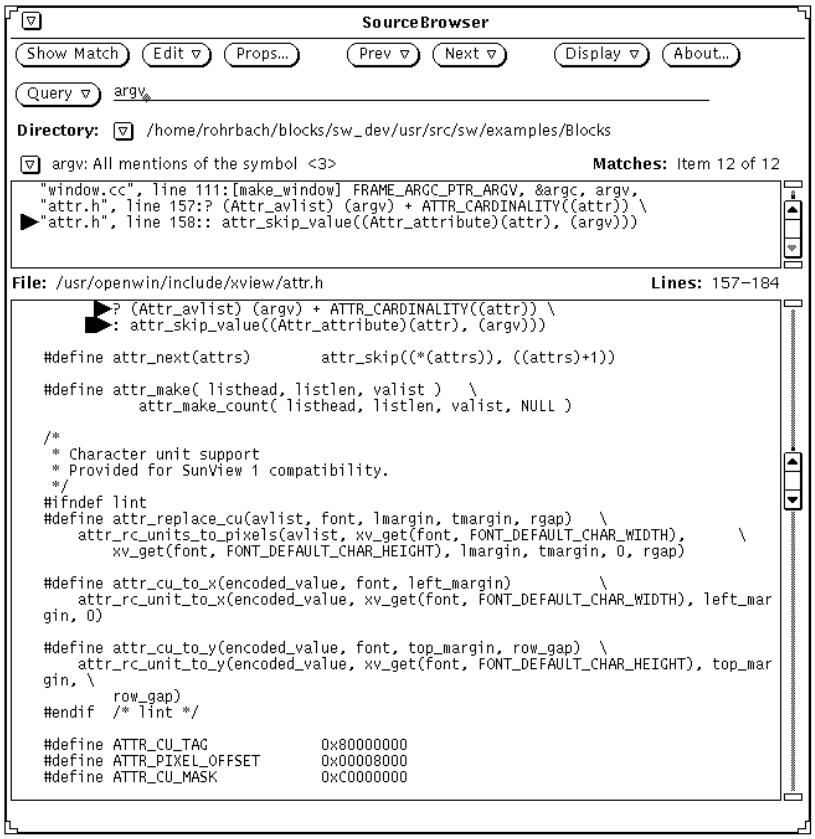
To view the previous match:



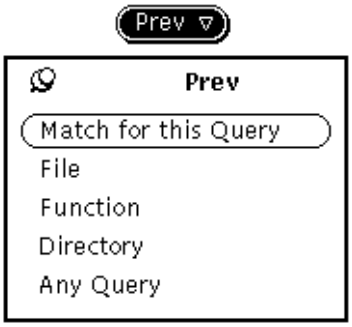
♦ **Click on the Prev (for Previous) button.**
SourceBrowser makes the previous match in the match pane the current match and displays its surrounding source code in the source pane.

In the following example, the user first issued a query on the symbol `argv`. The user then instructed SourceBrowser to display the previous match of `argv`, which it found in the `attr.h` file in the `xview` directory.

SourceBrowser scrolled to the bottom of the match list



To display the last match that occurs in the previous function, file, or directory:



♦ **Choose File, Function, or Directory from the Prev menu.**

In the following example, the user chose File from the Prev menu. SourceBrowser moved the current match to the last occurrence of argv in the `window.cc` file.

Current match is now in the file window.cc

4.4 Viewing a Selected Match

If you are browsing through the match list and you see a match that you want to display in the source pane, do the following:

- Click on the line in the match pane
- Click on Show Match

4.5 Moving to a Match From Any Query

Often, your file will contain matches from several queries. Matches from a previous query are marked with small black arrows in the source pane. You can move between the matches in your file regardless of the query to which the match belongs.

To view a match from any query:

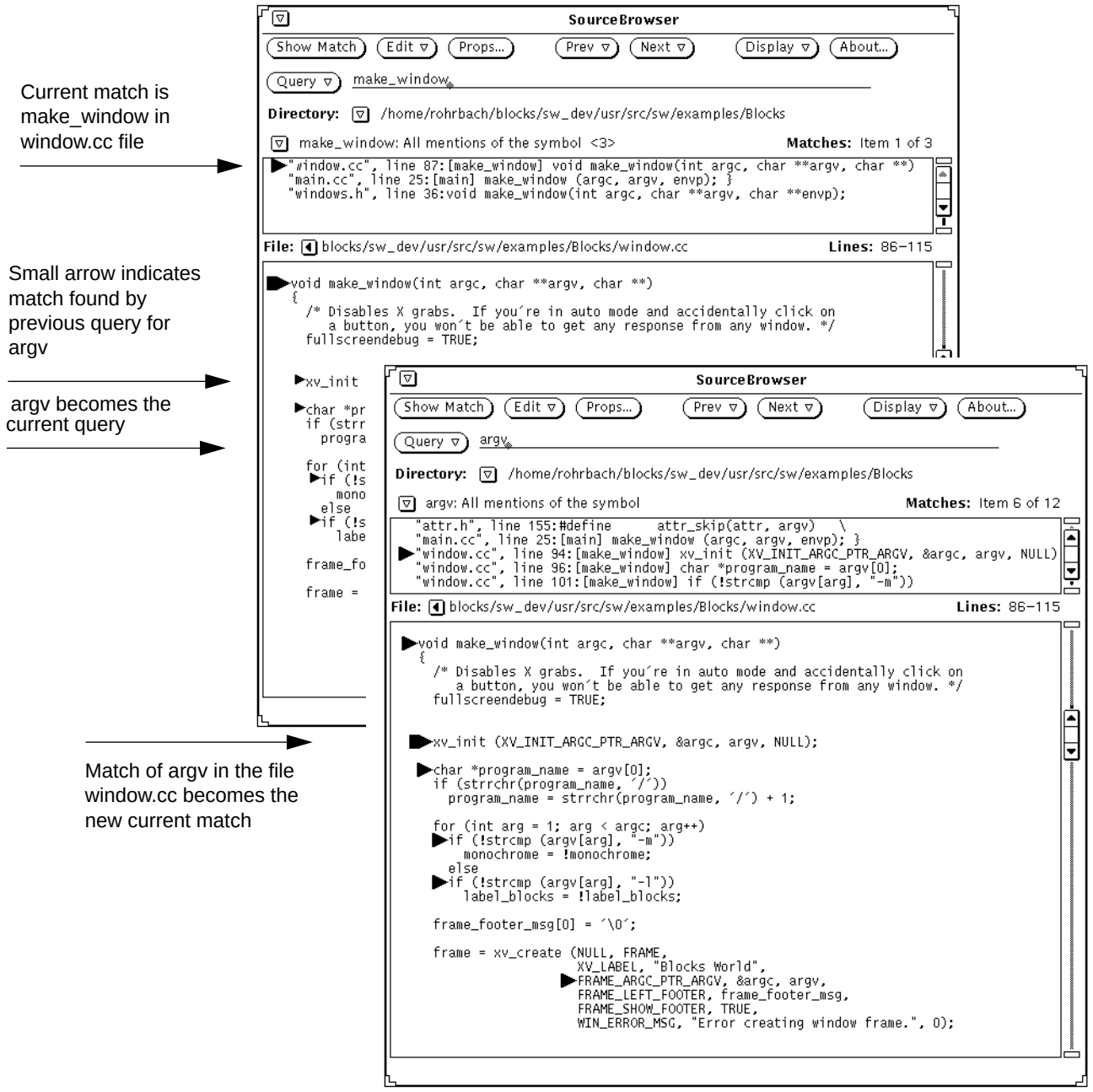
1. **Use the scroll bar in the source pane to scroll until the match you want to select is displayed.**
2. **Select the match by clicking anywhere on the line containing the match.**
3. **Click on the Show Match button.**
   This is equivalent to choosing Selected Match from the View menu. If the match was found by another query, then that query becomes the current query.

You can specifically instruct SourceBrowser to view the next or previous match from any query:

♦ **Choose Any Query from either the Next or Prev menu.**

For example, suppose you issue a query on `argv`, then issue a query on `make_window` in the Blocks program. The current match is the first match of `make_window` in the file `window.cc`.

When ypou choose Any Query from the Next menu, SourceBrowser makes the next occurrence of `argv` in the file `window.cc`, the current match.

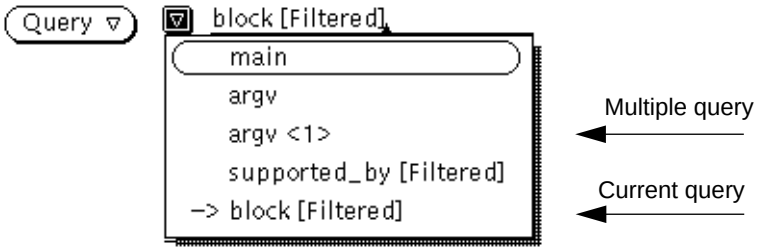

Current match is make_window in window.cc file
Small arrow indicates match found by previous query for argv
argv becomes the current query
Match of argv in the file window.cc becomes the new current match
SourceBrowser
Show Match
Edit
Props...
Prev
Next
Display
About...
Query
make_window
Directory:
/home/rohrbach/blocks/sw_dev/usr/src/sw/examples/Blocks
make_window: All mentions of the symbol <3>
Matches: Item 1 of 3
"window.cc", line 87:[make_window] void make_window(int argc, char **argv, char **)
"main.cc", line 25:[main] make_window (argc, argv, envp); }
"windows.h", line 36:void make_window(int argc, char **argv, char **envp);
File:
blocks/sw_dev/usr/src/sw/examples/Blocks/window.cc
Lines: 86-115
void make_window(int argc, char **argv, char **)
{
/* Disables X grabs. If you're in auto mode and accidentally click on
a button, you won't be able to get any response from any window. */
fullscreendebug = TRUE;
xv_init
char *pr
if (strr
progra
for (int
if (!s
mono
else
if (!s
labe
frame_fo
frame =
SourceBrowser
Show Match
Edit
Props...
Prev
Next
Display
About...
Query
argv
Directory:
/home/rohrbach/blocks/sw_dev/usr/src/sw/examples/Blocks
argv: All mentions of the symbol
Matches: Item 6 of 12
"attr.h", line 155:#define attr_skip(attr, argv) \
"main.cc", line 25:[main] make_window (argc, argv, envp); }
"window.cc", line 94:[make_window] xv_init (XV_INIT_ARGC_PTR_ARGV, &argc, argv, NULL)
"window.cc", line 96:[make_window] char *program_name = argv[0];
"window.cc", line 101:[make_window] if (!strcmp (argv[arg], "-m"))
File:
blocks/sw_dev/usr/src/sw/examples/Blocks/window.cc
Lines: 86-115
void make_window(int argc, char **argv, char **)
{
/* Disables X grabs. If you're in auto mode and accidentally click on
a button, you won't be able to get any response from any window. */
fullscreendebug = TRUE;
xv_init (XV_INIT_ARGC_PTR_ARGV, &argc, argv, NULL);
char *program_name = argv[0];
if (strrchr(program_name, '/'))
program_name = strrchr(program_name, '/') + 1;
for (int arg = 1; arg < argc; arg++)
if (!strcmp (argv[arg], "-m"))
monochrome = !monochrome;
else
if (!strcmp (argv[arg], "-l"))
label_blocks = !label_blocks;
frame_footer_msg[0] = '\0';
frame = xv_create (NULL, FRAME,
XV_LABEL, "Blocks World",
FRAME_ARGC_PTR_ARGV, &argc, argv,
FRAME_LEFT_FOOTER, frame_footer_msg,
FRAME_SHOW_FOOTER, TRUE,
WIN_ERROR_MSG, "Error creating window frame.", 0);

4.6 Moving Between Queries

During a SourceBrowser session, you will often issue several queries. SourceBrowser maintains a history of these queries, called *active queries*, in the abbreviated menu next to the Query button so you can easily move between them. SourceBrowser always enters the new query as the last item in the menu.

To return to a previous query:

- **Choose the desired query from the abbreviated menu next to the Query button.**



If the Query menu contains multiple entries for the same symbol, such as `argv`, `argv<1>`, and `argv<2>`, then you have issued multiple queries on that symbol.

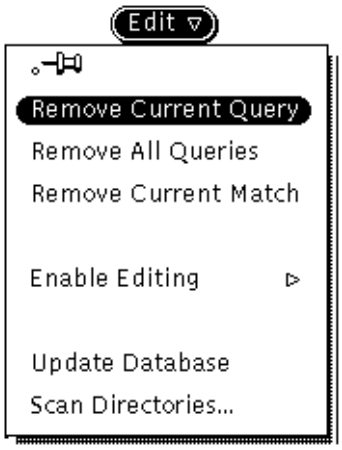
You can edit the list of active queries using the following items on the Edit menu (discussed in detail in Chapter 8, “Integrating Editing and Debugging”):

- Remove Current Query
- Remove All Queries

4.7 Removing Matches

You can remove unwanted matches from the match list. By removing unwanted matches, you can maintain a list of only the most important matches.

To remove a match:



- **Make the match you want to remove the current match, then choose Remove Current Match from the Edit menu.**
  The number of matches shown in the Matches field in the control area is reduced by one.

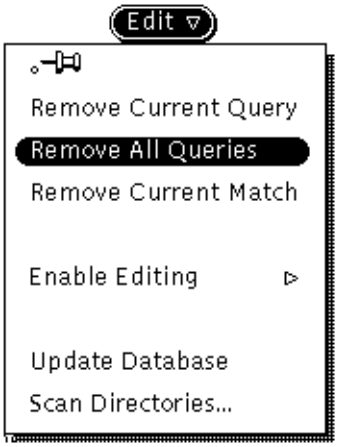
4.8 Removing Queries

When a query is no longer of interest, you can remove it from the list of active queries. This action frees up the memory SourceBrowser uses to store information about the query. The memory is reused by subsequent queries.

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

- **Click on the Edit button.**

To remove all queries from the list of active queries:



- **Choose Remove All Queries from the Edit menu.**

Graphing Functions and Classes 5

This chapter describes the CallGrapher and the ClassGrapher. This chapter is organized into the following sections:

Graphing Function Calls	*page 62*
Graphing Class Inheritance	*page 70*
Adding a Node to the Graph	*page 76*
Finding a Node in the Graph	*page 78*
Editing the Graph	*page 79*
Redoing the Layout	*page 81*
Printing the Graph	*page 81*
Querying a Function or Class	*page 81*
Displaying the Source of a Function or Class	*page 83*
Customizing the Call and Class Graphs	*page 85*
Quitting the CallGrapher and ClassGrapher	*page 92*

The first section provides instructions specific to graphing functions and the following section provides instructions specific to graphing classes. The remaining sections provide instructions that can be applied to either the call graph or the class graph.

5

5.1 Graphing Function Calls

The CallGrapher helps you visualize and understand function relationships in ANSI C, FORTRAN, Pascal, and C++ programs. It identifies the functions that either call or are called by one or more selected functions.

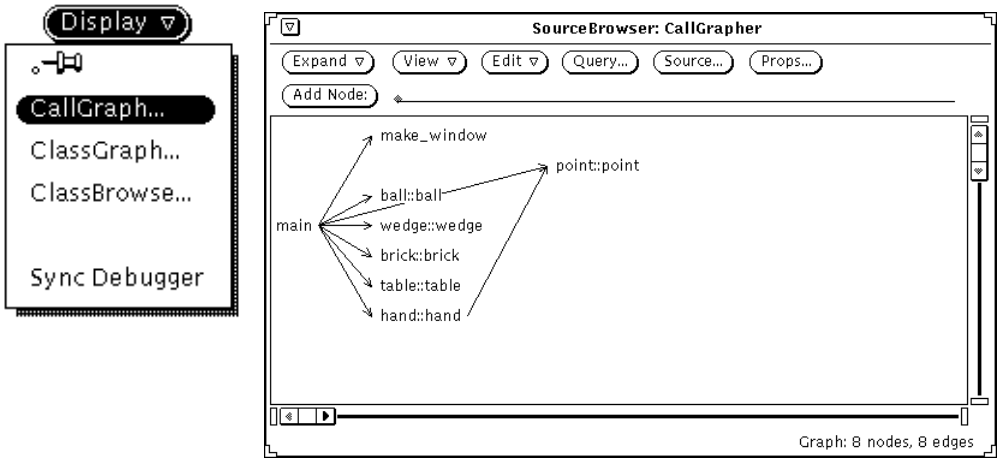
To graph the relationships of the functions in your program:

1. **If there is more than one program in your database, then set the Focus to the program whose functions you want to graph.**
   Choose Program from the Focus submenu, then deactivate the programs you do not want to search using the Deactivate menu in the Focus window. If you do not focus on a single program, then the CallGrapher will display confusing results - combining functions from several programs.

2. **Select the name of the function you want to display.**
   The function name can be a name you have typed and selected in the SourceBrowser Text field or an identifier you selected in another window. If you are unsure of where to start, then begin by selecting a function central to your program. For example, if your program is written in ANSI C or C++, then you might graph the symbol `main`.

3. **In the SourceBrowser base window, choose CallGraph from the Display menu.**

SourceBrowser graphs the selected function in the CallGrapher window. In this example, the user graphed the functions called by `main`.



The call graph consists of nodes and edges. A node represents a function. An edge represents a function call.

If there is a curved or back edge in the graph, then your program contains a recursive cycle. For example, the program may have a function A that calls function B which calls function A.

Note – There is an exception in the case of hand-editing. If you hand-edit a graph moving node A to the right of node B (given A -> B), a curved edge is created with the edge coming from the right side of A and entering the left side of B. Thus, curved edges only guarantee recursion *before* hand-editing a graph.

A function node that is to the left of (for a horizontal layout) or above (for a vertical layout) a given node indicates it is calling the node. The connecting edge between two such nodes represents the function call.

To call up an empty CallGrapher window:

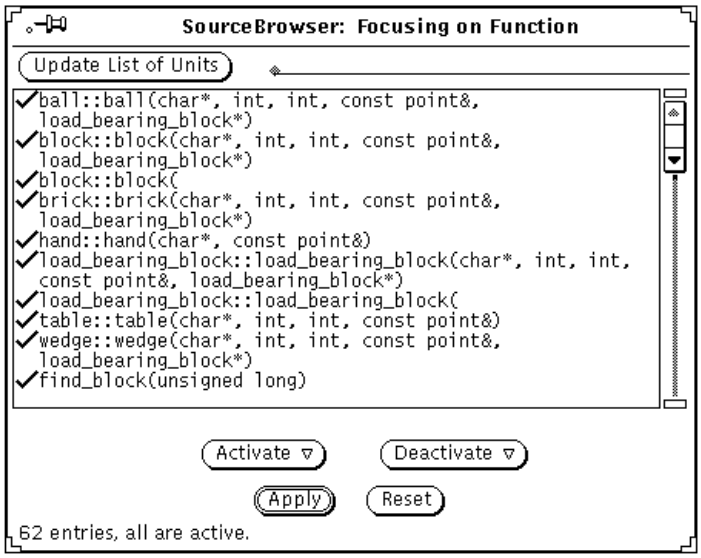
- **Choose CallGrapher from the Display menu without making a selection.**

To add a node to the graph:

- **Type a function name in the CallGrapher Text field and press return.**
  See, **"**Adding a Node to the Graph,**"** in this chapter, for more information.

To list all the functions in your program:

- **Use the Focus menu in the SourceBrowser base window.**
  See "Focusing A Query" on page 129 for more information. Here is the Focus function window for the Blocks program.



5.1.1 Graphing Overloaded and Virtual Functions in C++ Programs

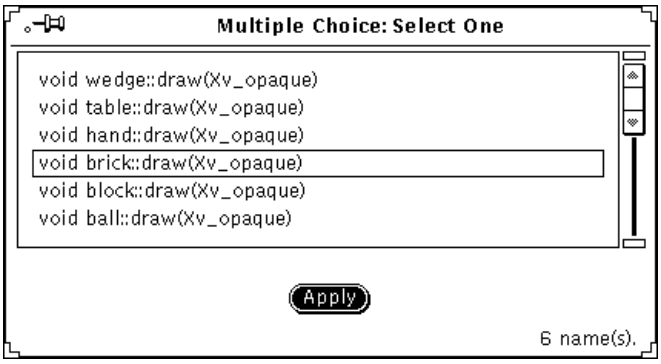
CallGrapher graphs overloaded functions as follows:

- When you attempt to graph an overloaded function, SourceBrowser activates the Multiple Choice window. To continue graphing, select an instance of the overloaded function, then click on the Apply button.
- You can include both the function name and its argument-type signature in the node name so that you can distinguish among the overloaded functions in the graph. To do so, set Function Names to Long in the CallGrapher Properties window.
- C++ member function names are always shown with their class-scope modifier "class-name::" (as in `block::put_on`) whether or not argument-type signatures are displayed.

CallGrapher graphs virtual functions as follows:

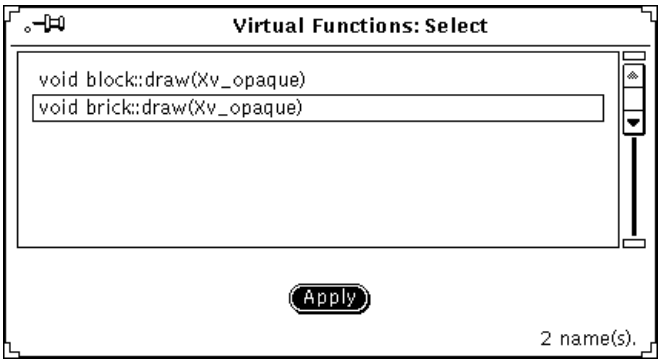
- A virtual function named `draw` appears as `virtual draw` in the graph.
- A standard function named `box` that is associated with one or more virtual functions appears as `<box>`.
- When you attempt to graph a virtual function, and there is more than one virtual function in the program, SourceBrowser activates the Virtual Functions popup window. You can select more than one of the virtual functions listed in the window; the scrolling list is nonexclusive.
- If you expand a standard function that has one or more associated virtual functions in the called by direction, CallGrapher pops up a notice, asking if you want to see the virtual or standard functions that call it.

In this example, the user first typed `draw` in the SourceBrowser Text field, then chose CallGraph from the Display menu. Because the function `draw` is an overloaded function, SourceBrowser activated the Multiple Choice window. To continue graphing, the user chose `brick::draw`, then clicked on the Apply button.

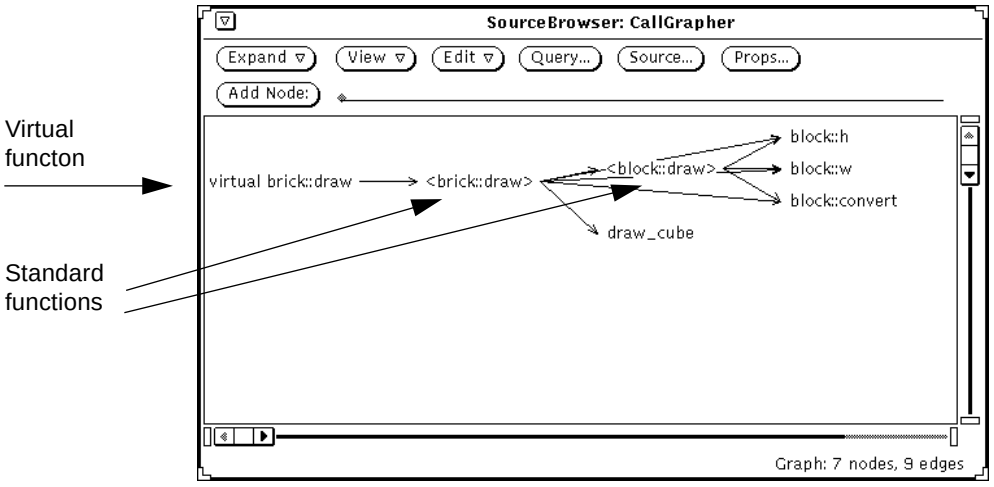


The node `brick::draw` represents a virtual function. SourceBrowser popped up a notice asking whether the user wanted to expand the virtual or standard calls. The user chose virtual.

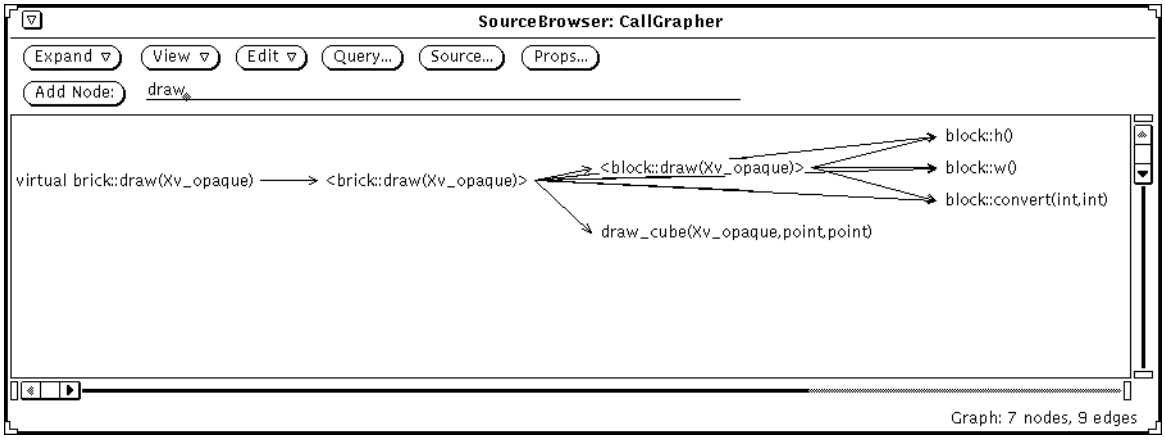
SourceBrowser next activated the Virtual Functions popup window. This window lists two virtual functions in the Blocks program. The user chose `brick::draw`, then clicked on the Apply button.



The resulting graph includes one virtual and two standard functions. If the user attempts to expand the standard function `<brick::draw>` in the called by direction, the CallGrapher pops up a notice asking if the user wants to see the virtual or standard functions that call it. See the next section for more information on expanding functions.



Finally, the user set Function Names to Long in the CallGrapher Properties window. The node names now include both the function name and its argument-type signature in the node name

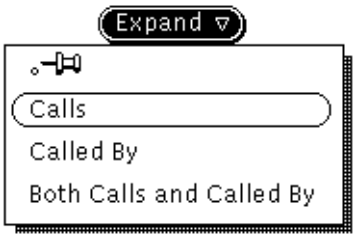


5.1.2 Expanding Functions

The initial call graph includes *n* level of functions that either call or are called by the selected function, where *n* is the expansion depth set in the CallGrapher Properties window. You can add functions that call or are called by any node in the graph. This process is known as *expansion*. The difference between expanded and unexpanded nodes is that an unexpanded node might not show all of the incoming or outgoing edges. To insure that all edges are shown, use the expanded mode. In the graphs, expanded nodes are shown in Roman font and unexpanded nodes are shown in Italics (Bold for Asian versions).

To expand the functions of a node in the graph:

♦ **Select the node or nodes you want to expand, then choose an item from the Expand menu.**



The menu items are:

Calls—Expand the call graph in the forward, or calls, direction from the selected nodes.

Called by—Expand the call graph in the backward, or called-by direction from the selected nodes.

Both Calls and Called by—Expand the call graph in both forward and backward direction from the selected nodes.

The Expand operations add nodes to the graph and all edges between both new and old nodes. The Expand operations work only on selected nodes.

The expansion depth represents the distance from the selected nodes within which the Grapher adds nodes and edges to the graph. For example, if you set the expansion depth to two, then the Grapher adds all nodes and edges two levels away from the selected node, as shown in Figure 5-1.

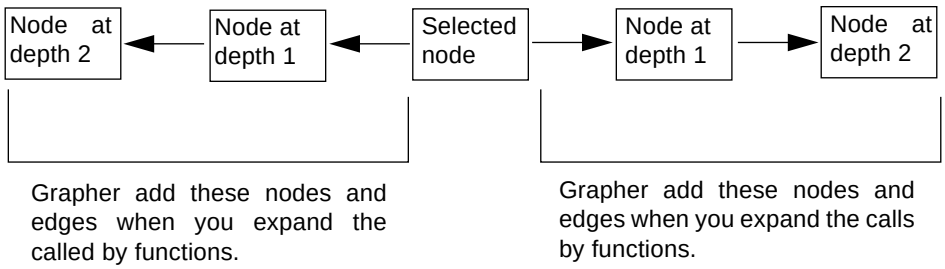


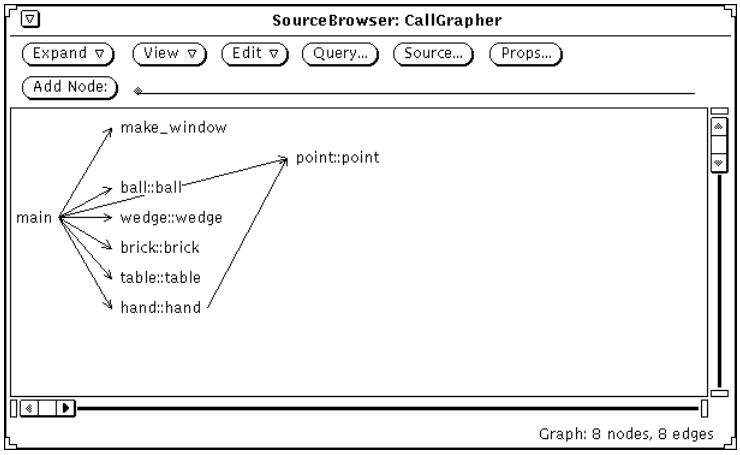
Figure 5-1 Expansion Depth of Two in CallGrapher

To change the expansion depth:

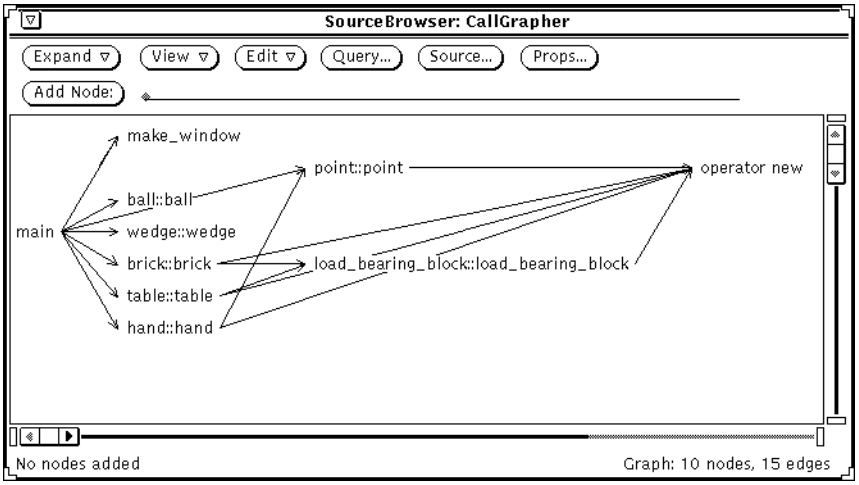
- **Open the Grapher Properties window and click the up or down arrow in the Expansion Depth field. Then click on the Apply button.**
  See “Customizing the Call and Class Graphs” on page 85 for more information.

In this example, the user expanded the graph in the calls direction for the nodes `brick`, `table`, and `hand`. The expansion depth is set to one.

Graph before expanding brick, table, and hand



Graph after expansion



5.2 Graphing Class Inheritance

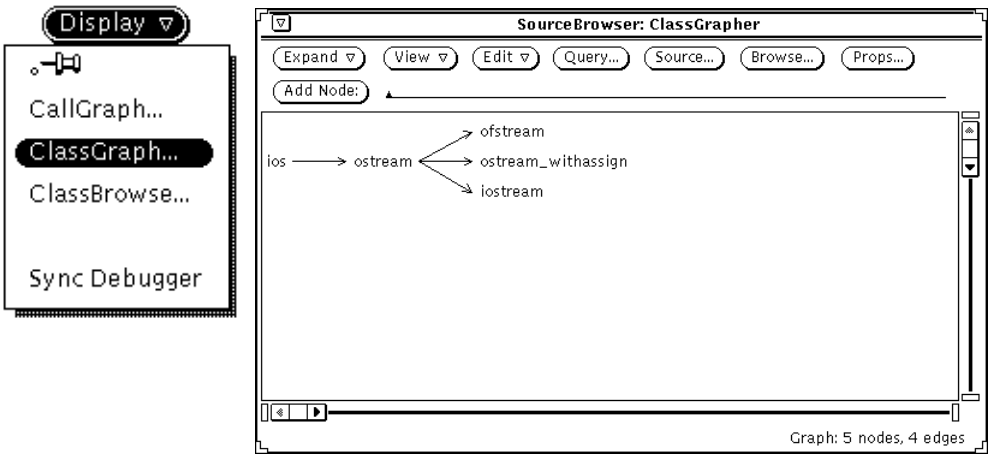
The ClassGrapher helps you visualize and understand C++ class hierarchies. It identifies the derived classes, base classes, and class inheritances of the program.

To graph the class hierarchy in your program:

1. **If there is more than one program in your database, then set your Focus to the program whose class inheritance you want to graph.**
   Choose Program from the Focus submenu, then deactivate the programs you do not want to search using the Deactivate menu in the Focus window. If you do not focus on a single program, then ClassGrapher displays a graph that represents all programs in the database.

2. **Select the name of the class you want to display.**
   The class name can be a name you have typed and selected in the SourceBrowser Text field or an identifier you selected in another window.

3. **In the SourceBrowser window, choose ClassGraph from the Display menu.**
   SourceBrowser displays the graph in the ClassGrapher window. In this example, the user graphed the classes related to `ostream`



In the class graph, a node represents a class type. A class node that is to the left of (for a horizontal layout) or above (for a vertical layout) a given node indicates it is a parent of the node. The connecting edge between two such nodes represents the class inheritance.

To call up an empty ClassGrapher window:

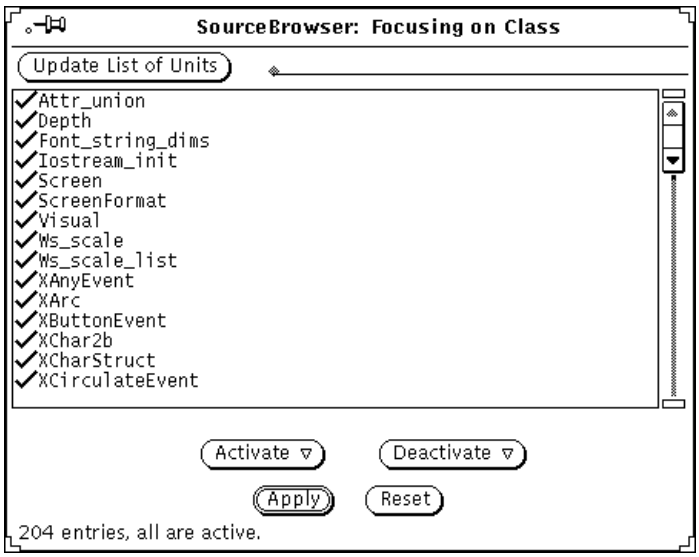
- **Choose ClassGrapher from the Display menu without making a selection.**

To add a node to the graph:

- **Type a class name in the ClassGrapher Text field and press return.** See the **"**Adding a Node to the Graph,**"** in this chapter for more information.

To list all the classes in your program:

- **Use the Focus menu in the SourceBrowser base window.** See "Focusing A Query" on page 129 for details. Here is the Focusing on Class window for the Blocks database.

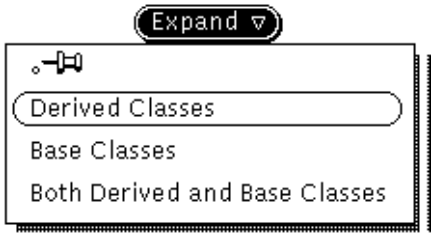


5.2.1 Expanding Classes

The initial class graph includes all base and derived classes *n* levels away from the selected class, where *n* is the expansion depth set in the ClassGrapher Properties window. You can add the base and derived classes of any node in the graph. This process is known as *expansion*.

To expand the classes in the class graph:

- **Select the node or nodes you want to expand, then choose an item from the Expand menu.**



The menu items are:

Derived Classes—Expand the class graph in the forward, or derived class, direction from the selected nodes.

Base Classes—Expand the class graph in the backward, or base class direction from the selected nodes.

Both Derived and Base Classes—Expand the class graph in both forward and backward direction from the selected nodes.

The Expand operations add nodes to the graph and all edges between both new and old nodes. The Expand operations work only on selected nodes.

The expansion depth represents the distance from the selected nodes within which the Grapher adds nodes and edges to the graph. For example, if you set the expansion depth to two, then the Grapher adds all nodes and edges two levels away from the selected node, as shown in Figure 5-2.

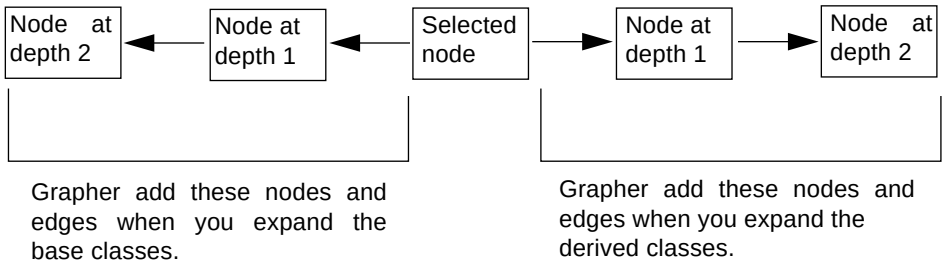


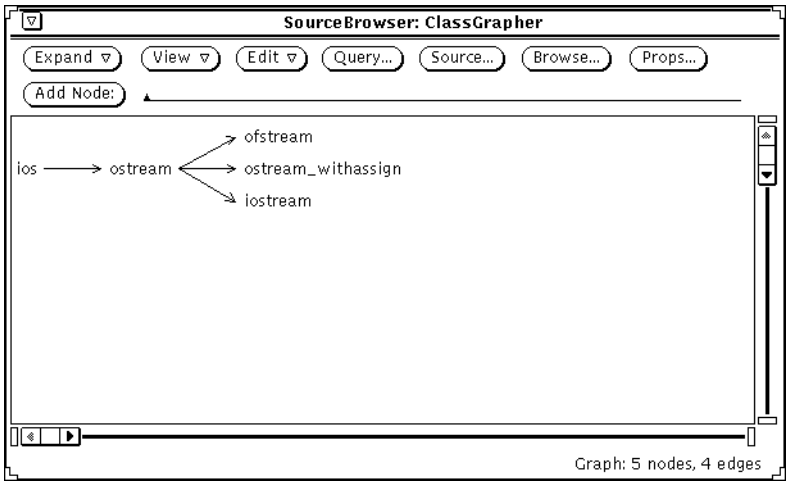
Figure 5-2 Expansion Depth of Two in ClassGrapher

To change the expansion depth:

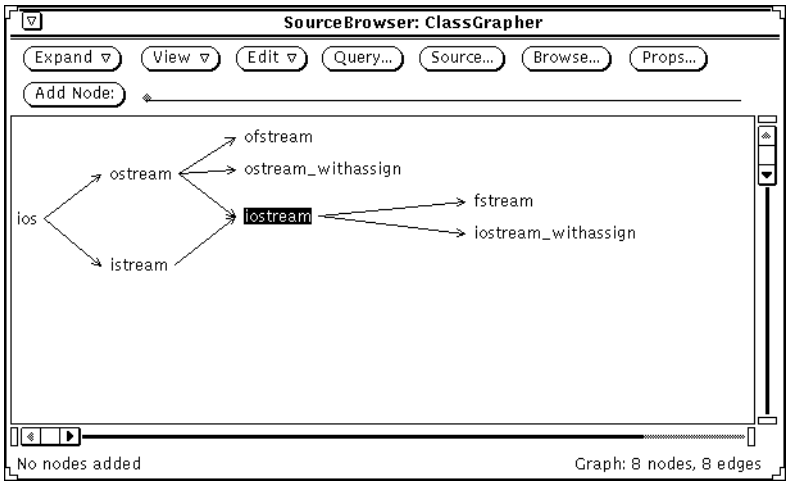
- **Open the Grapher Properties window and click on the up or down arrow in the Expansion Depth field. Then, click on the Apply button.**
  See “Customizing the Call and Class Graphs” on page 85 for more information.

In this example, the user expanded both the derived and base classes in `iostream`. The expansion depth is set to one.

Graph before expanding istream

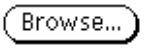


Graph after expansion



5.2.2 Browsing Classes

To browse a class from the ClassGrapher window:

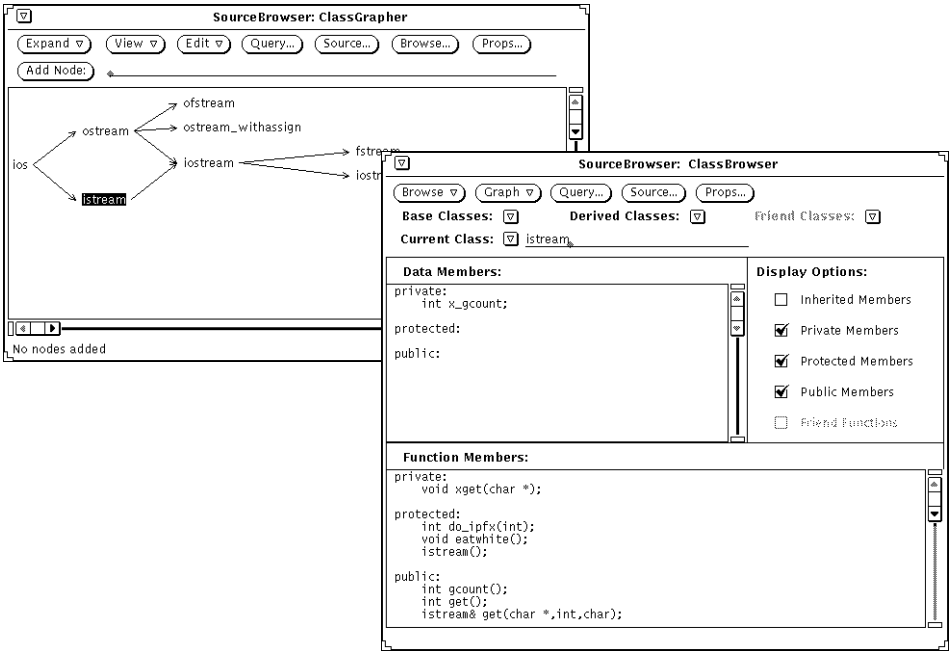


- **Select the node you want to browse, then click on the Browse button.**
  If you select more than one node, then ClassGrapher browses the last node selected.

When you browse a class, ClassGrapher behaves as follows:

- ClassGrapher activates a new ClassBrowser window if no ClassBrowser is running.
- ClassGrapher displays the class in the existing ClassBrowser window if one ClassBrowser is already running.
- ClassGrapher displays the class in the last ClassBrowser activated if multiple ClassBrowsers are running.

In this example, the user selected the node `istream`, then clicked on the Browse button.



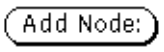
As a shortcut, you can browse a class by simply double-clicking on its node. To set up double-click:

- **Open the ClassGrapher Properties window and set Double-click to Browse. Then, click on the Apply button.**

See Chapter 6, **"**Browsing Classes,**"** for more information on the ClassBrowser.

5.3 Adding a Node to the Graph

To add a node in either the call or class graph:

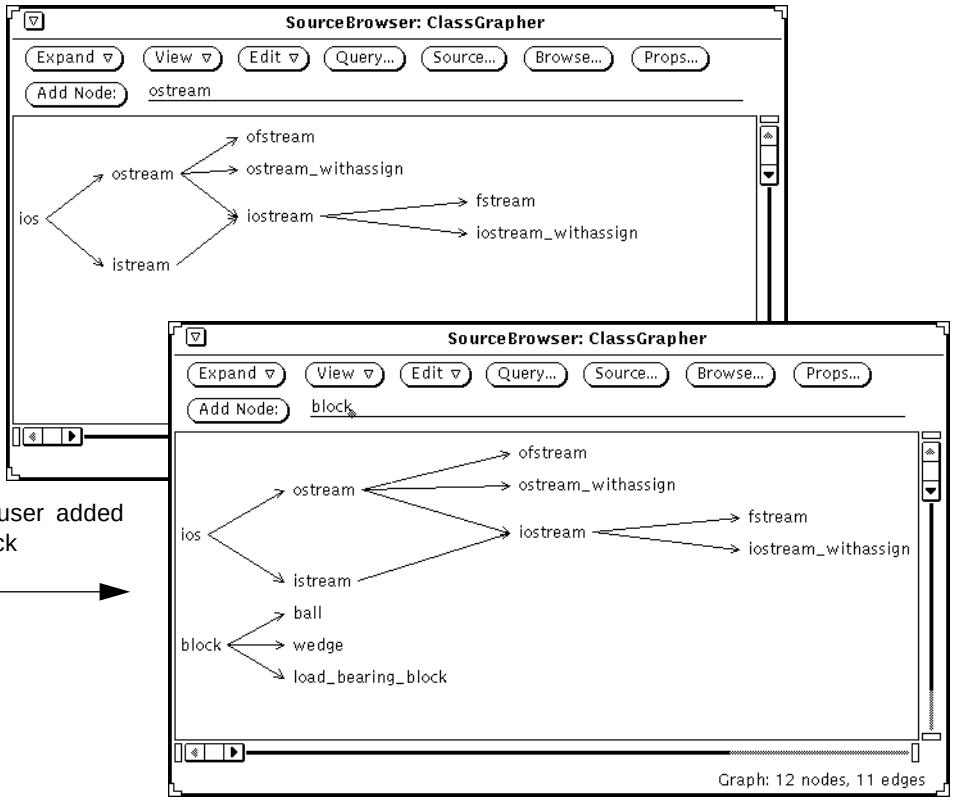


- **Select the name of the node you want to graph, then click on the Add Node button.**
  The node name can be a name you have typed in the Text field, or a name you have selected in the SourceBrowser or another window.

The Grapher adds the node to the graph and expands the graph around the added node. The Grapher also adds all edges between new and old nodes.

In this example, the user began with a graph that showed the derived classes of `ios`. The user then typed and selected `block` in the Text field and clicked on the Add Node button. The Grapher now has two separate graphs.

Graph of ios



Graph after user added the class block

The following is important additional information about adding a node to the graph:

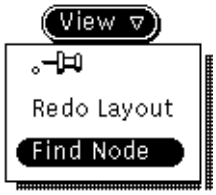
- When you use the Text field to add a node, you can simply type the node name in the Text field and press Return.
- If the added node does not have any connections to the existing graph, then the Grapher adds the node as a separate graph.
- If the node already exists in the graph, then the Grapher scrolls the graph so that the node is in the center of the display.

- If you set Selection to **Moved to Show Node** in the CallGrapher or ClassGrapher Properties window, then the Grapher highlights the node you add to the graph.
- If you are graphing a C++ program and you ask the CallGrapher to add an overloaded or virtual function, then SourceBrowser displays the Multiple Choices window with a list of all available choices of functions. Click on the desired function name in the Multiple Choices window, then click on the Apply button. The CallGrapher adds the node to the graph.

5.4 Finding a Node in the Graph

In large graphs, it is not always easy to locate a node in the graph. The Grapher provides a Find node operation to help you locate the node you are looking for.

To find a node in the graph:

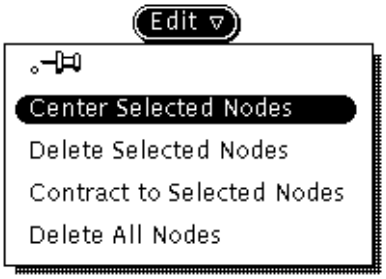


♦ **Select the name of the node you want to find, then choose Find Node from the View menu.**
The node name can be a name you have typed and selected in the Text field, or a name you have selected in the SourceBrowser or any other window.

If the Grapher finds the node, it repositions the graph, moving the node near the center of the window. If the Grapher Selection property is set to Move to Show Node, then the Grapher highlights the node.

5.5 *Editing the Graph*

To remove nodes and edges:



- **Select the node or nodes, then choose an item from the Edit menu.**

The menu items are:

Center Selected Nodes—Contracts, then expands, the selected nodes.

Delete Selected Nodes—Remove the selected nodes and the edges connecting the nodes with the rest of the graph.

Contract to Selected Nodes—Remove all nodes and edges from the graph that are beyond the contraction radius from the selected nodes. For example, if you set the contraction radius to two, the Grapher removes all nodes and edges at radius three or more from the selected node when you contract the graph, as shown in Figure 5-3. You set the contraction radius in the Properties window. The default radius is one.

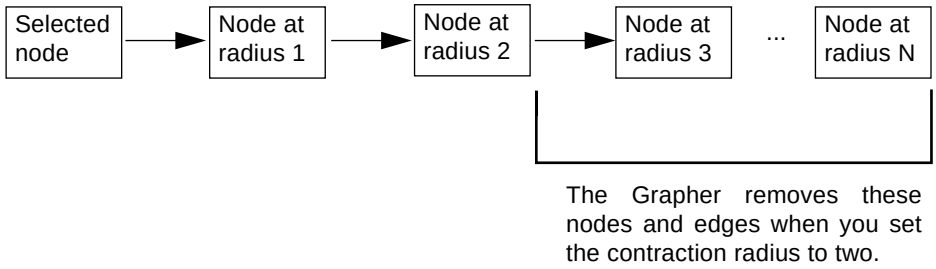
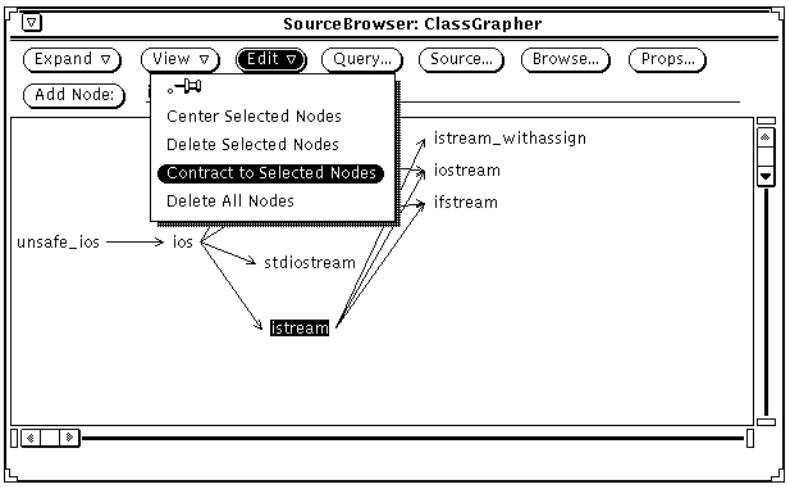


Figure 5-3 Contraction Radius of Two

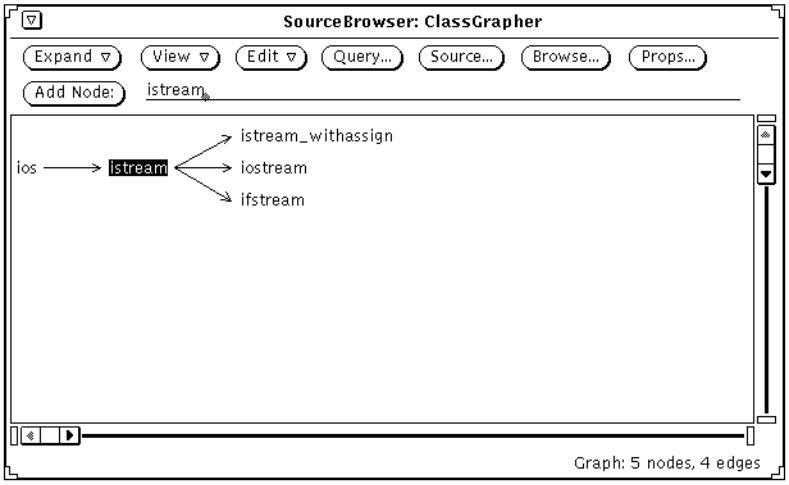
Delete All Nodes—Remove all nodes and edges from the graph, leaving an empty grapher window. You do not need to select a node before you choose this item.

In this example, the user contracted the graph around `istream`. The contraction radius is one. The resulting graph shows that `istream` has one base and three derived class.

Graph before contraction



Graph after user contracted around istream

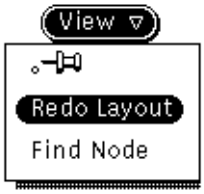


5.6 Redoing the Layout

The Grapher attempts to retain the position of the nodes as you edit the graph. However, if you have deleted a lot of nodes, then you may want to reposition the nodes so that the graph is easier to read.

To automatically reposition the nodes in the graph:

- **Choose Redo Layout from the View menu.**



To manually reposition the nodes:

- **Select the node or nodes you want to move, then drag them to the new position.**

5.7 Printing the Graph

There is no direct support for printing a graph. However, the graph can be printed by using the `snapshot` tool to capture the raster image of the graph window. See the `snapshot(1)` man page for details.

If the image is captured on a color monitor, it is captured at the monitor's color-depth. This can result in a large image and consequnetly a longer print time. A tool, such as `imagetool(1)` on Solaris 2, can be used to convert the image to black and white whic will reduce the file size and print time.

5.8 Querying a Function or Class

You can issue a query on any node in the call or class graph. SourceBrowser displays the results of the query in the SourceBrowser base window.

To issue a query on a node:

♦ **Select the node or nodes you want to query, then click on the Query button in the Grapher window.**
If you select more than one node, then SourceBrowser performs successive queries

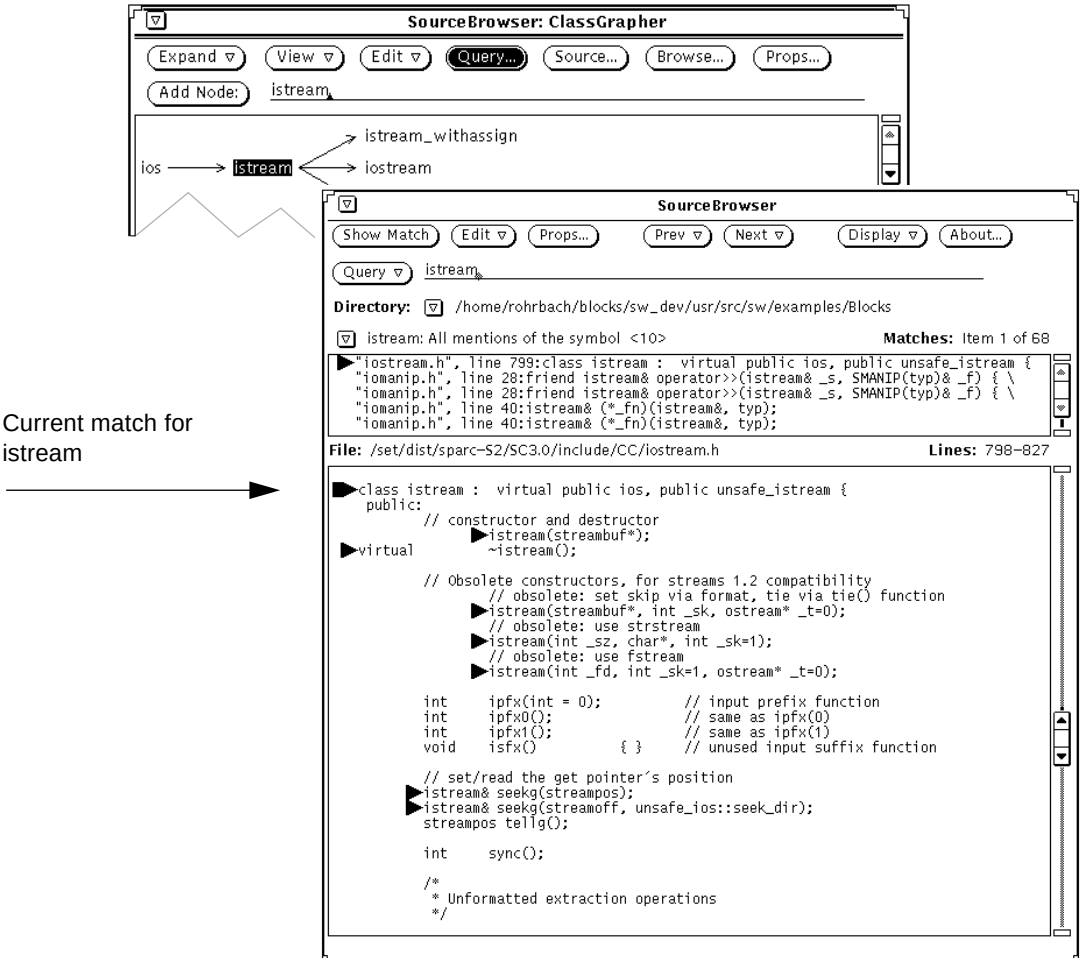
You can also issue a query by double-clicking on the node. To set up double-click:

♦ **Open the Grapher Properties window and set Double-click to Query. Then, click on the Apply button.**

When you issue a query on a node, SourceBrowser behaves as follows:

- If the SourceBrowser base window is in the background, then SourceBrowser brings it to the foreground.
- If the window is closed to an icon, then SourceBrowser opens it.
- If you previously quit the window, then SourceBrowser reopens it.

In this example, the user issued a query on the node `istream`.



5.9 *Displaying the Source of a Function or Class*

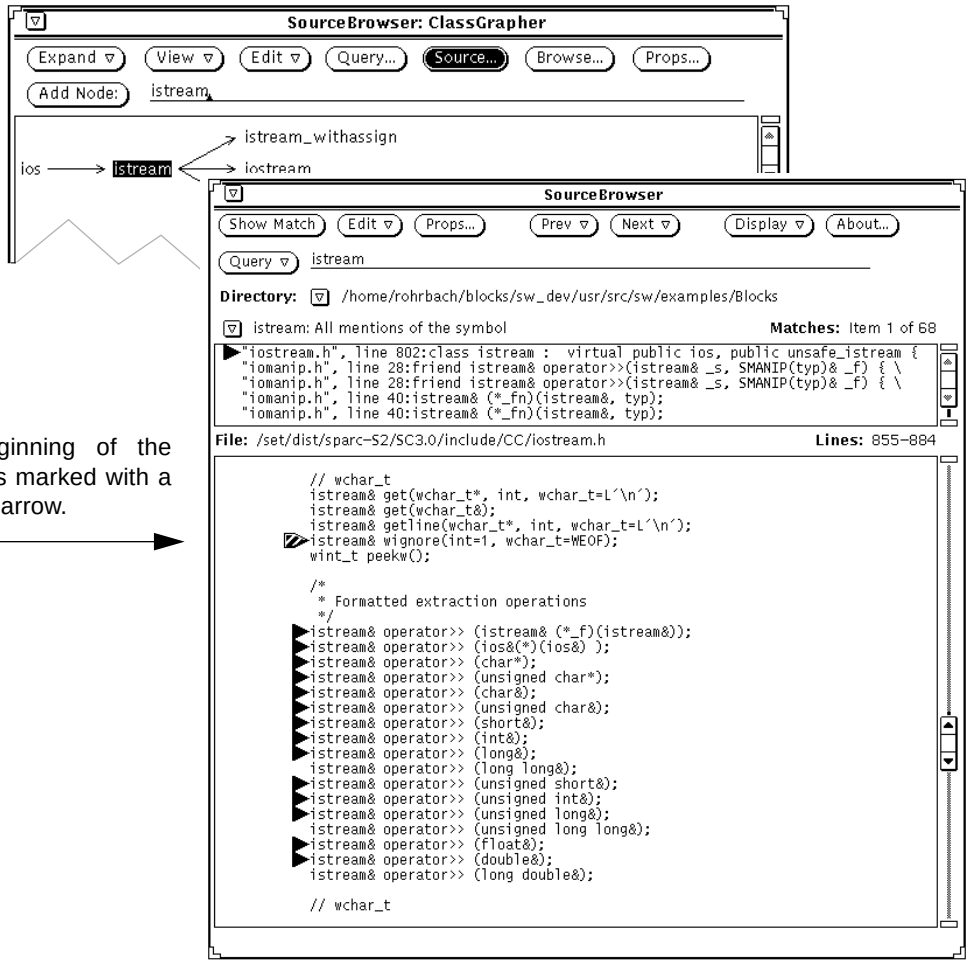
You can display the source code of any node in the call or class graph. You can display the source in either the SourceBrowser or the Debugger window.

5.9.1 Displaying Source in SourceBrowser

To display the source of a node in the SourceBrowser base window:

1. **Set Show Source In to Browser in the Grapher Properties window.** This setting is the default.
2. **Select the node, then click on the Source button in the Grapher window.** If you select more than one node, then the Grapher displays the source of the last node you selected.

In this example, the user displayed the source for the node istream.



The beginning of the source is marked with a hatched arrow.

You can also display the source of a function or class by double-clicking on its node. To set up double-click:

♦ **Open the Grapher Properties window and set Double-click to Query.**

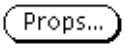
When you display the source of a node, SourceBrowser behaves as follows:

- If the SourceBrowser base window is in the background, then SourceBrowser brings it to the foreground.
- If the window is closed to an icon, then SourceBrowser opens it.
- If you previously quit the window, then SourceBrowser reopens it.

5.10 Customizing the Call and Class Graphs

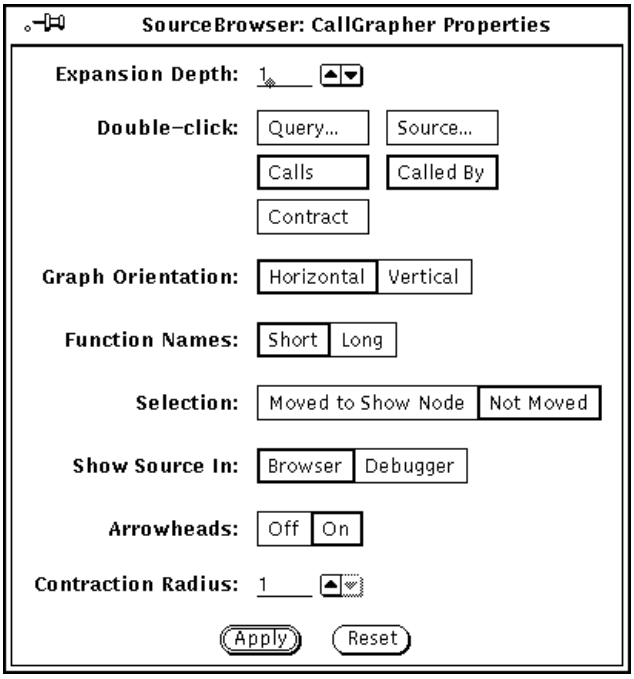
The Properties window lets you customize the properties associated with the graph.

To open the CallGrapher Properties window:

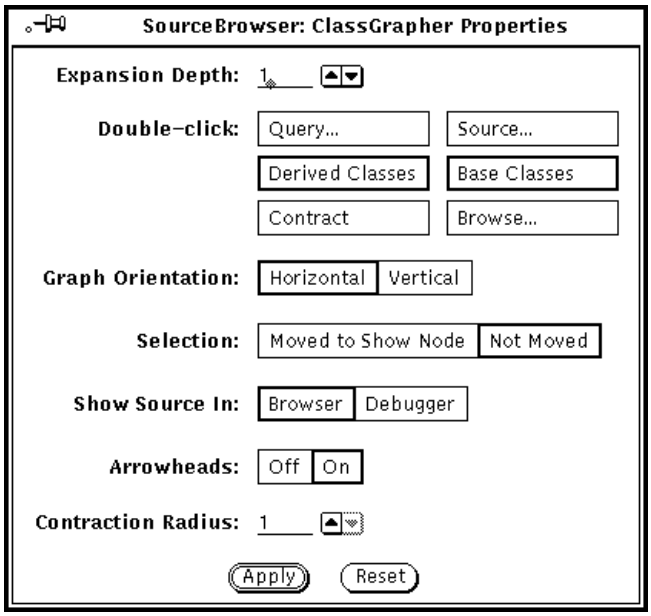


♦ **Click on the Props button in the CallGrapher window.**

Here is the CallGrapher Properties window.



And here is the ClassGrapher Properties window.



To set a property:

- **Click on the setting(s) of interest, then click on the Apply button.**
  Any changes you make to the Properties window are retained in the `.SBdefaults` file in your home directory. All subsequent Grapher sessions use the default property settings in the `.SBdefaults` file.

To discard any changes you have made, but have not yet applied:

- **Click on the Reset button.**
  SourceBrowser returns the properties to their last applied values. The Properties window remains open so that you can make further changes.

Following are descriptions of the Grapher properties.

Expansion Depth

The expansion depth represents the distance from the selected nodes to which the Grapher adds nodes and edges to the graph. The Grapher applies the expansion depth when you use either the Expand or Add Node buttons. The expansion depth only affects the nodes added to the graph.

You can either type the expansion depth in the text field or click the up or down arrows to the desired value. The minimum expansion is zero; in this case, you insert only the selected node. The default expansion is 1.

The larger the expansion depth, the more likely it is that a large number of nodes will be added to the graph during an expansion. Grapher operations on very large graphs can be slow

Double-click

This property tells the Grapher the operation(s) to perform when you double-click on a node. These settings are not mutually exclusive. The operations can also be performed by buttons and menus in the Grapher control area.

Query—Issue a query on the node.

Source—Show the node definition and its surrounding source code in either the SourceBrowser or the Debugger window.

Calls—CallGrapher only. Expand the functions that are called by the node. This setting is on by default.

Called By—CallGrapher only. Expand the functions that call the node. This setting is on by default.

Derived Classes—ClassGrapher only. Expand the classes derived from the node. This setting is on by default.

Base Classes—ClassGrapher only. Expand the base classes of the node. This setting is on by default.

Contract—Trim the graph around the selected node or nodes. All nodes and edges beyond the contraction radius from the selected nodes are removed from the graph. If both the expand and contract operations are set, then the Grapher performs the expansion first, then the contraction.

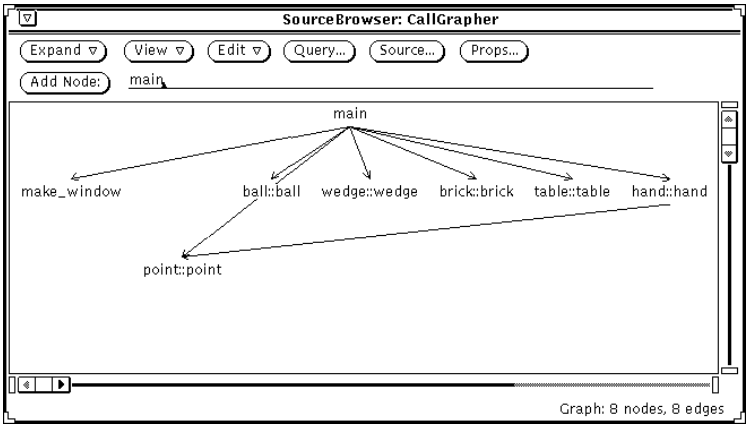
Browse—ClassGrapher only. Browse the node in the ClassBrowser window.

Graph Orientation

Graph Orientation determines the direction of the graph layout.

Horizontal—Display the nodes from left to right. This setting is the default.

Vertical—Display the nodes from top to bottom. The following graph is an example of a vertical layout.

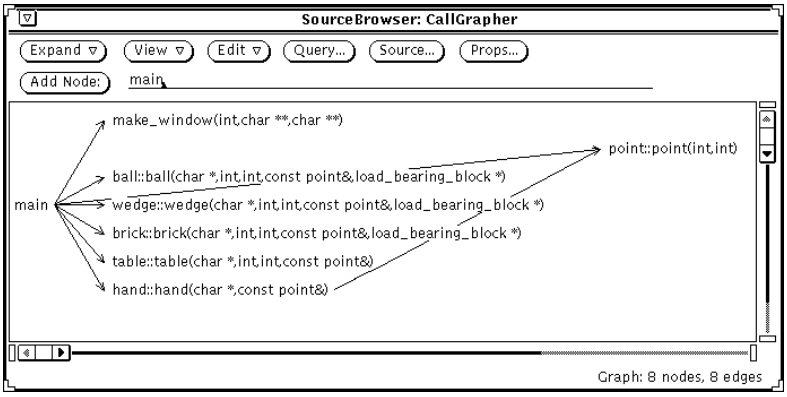


Function Names *(CallGrapher only)*

Function Names applies to C++ function names.

Short—Use the function name only as the node name. This is the default.

Long—Include both the function name and its argument-type signature in the node name. Turn on this setting when you need to distinguish among overloaded function names in C++ programs.



Selection

Selection determines whether or not the Grapher moves the selection during a find node, add node, or show source operation.

Moved to Show Node—Move the selection to the node you just added, found, or showed the source of.

Not Moved—Do not move the selection to the node you just operated on. This setting is the default.

Show Source In

Show Source In tells the Grapher where to display the source of a node when you click on the Source button.

Browser—Display the source code of the selected node in the SourceBrowser base window.

Debugger—Display the source in the Debugger window.

Arrowheads

Arrowheads determines the appearance of the edges in the graph.

Off—Do not put an arrowhead at the end point of each edge.

On—Put an arrowhead at the end point of each edge to indicate its direction. This setting is the default. The arrowheads do not add any additional information to the graph. An incoming edge always appears on the left or on the top of the node. An outgoing edge is always to the right or bottom of the node.

Contraction Radius

The Contraction Radius represents the distance from the selected node(s) beyond which the Grapher removes nodes and edges when you contract the graph. Click the up or down arrow buttons to the desired value or type a value in the text field. The minimum radius is 1 node. The default radius is 1 node.

5.11 Quitting the CallGrapher and ClassGrapher

Note – Remember that it is also possible to close the window to an icon. Once you dismiss the window, you cannot retrieve the graph. To view the graph again, you must activate a new window from either the SourceBrowser or ClassBrowser.

To quit the CallGrapher or Class Grapher:

♦ **Choose Quit from the Grapher menu.**

Browsing Classes 6

Use ClassBrowser to browse through C++ source code and libraries. You can view the defined classes and their data and function members. By navigating through the classes in the source code and libraries, you can understand how the classes were defined and used, and reuse them if possible.

This chapter is divided into the following sections:

Browsing the First Class	*page 94*
Browsing Subsequent Classes	*page 97*
Viewing Data and Function Members	*page 101*
Viewing Virtual and Inline Functions	*page 103*
Traversing the Class Hierarchy	*page 104*
Revisiting Previously Browsed Classes	*page 105*
Graphing Class Inheritance	*page 106*
Listing Class Types	*page 107*
Querying for Classes and Functions	*page 110*
Displaying the Source of Classes and Functions	*page 112*
Customizing ClassBrowser	*page 114*
Quitting ClassBrowser	*page 116*

6

6.1 Browsing the First Class

You can activate ClassBrowser from either the SourceBrowser or ClassGrapher window.

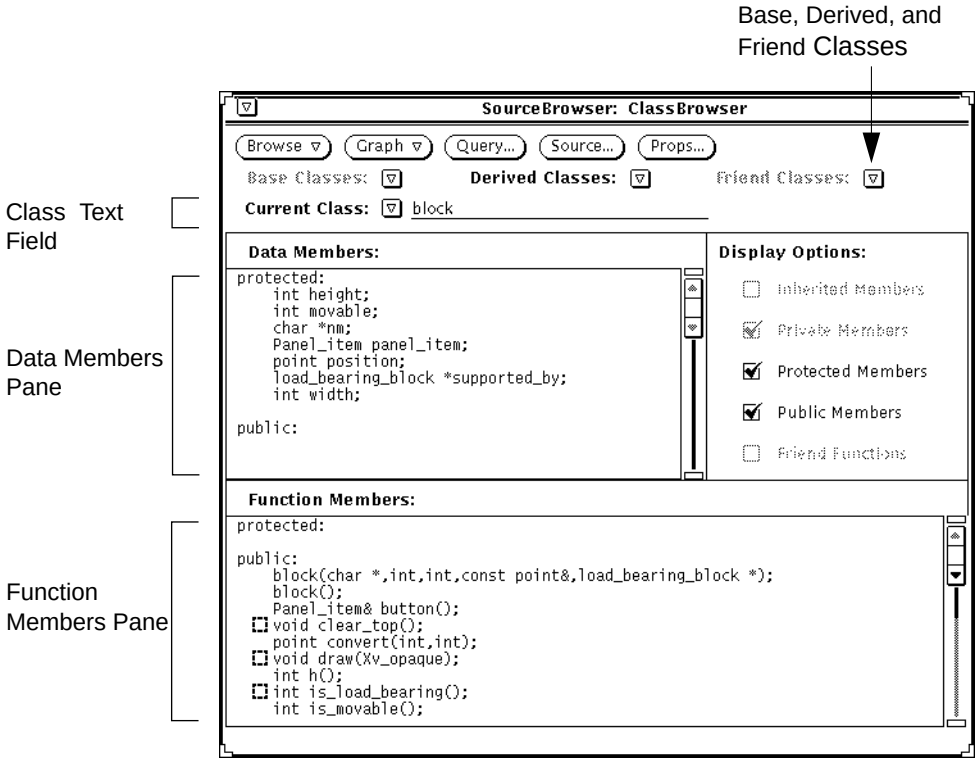
6.1.1 Activating ClassBrowser From SourceBrowser

To activate ClassBrowser from SourceBrowser:

1. **Select the name of the class that you want to browse.**
   The class name can be a name you have typed in the Text field or a class name you selected in a SourceBrowser pane or in another window.

2. **Choose the ClassBrowse item in the SourceBrowser Display menu.**

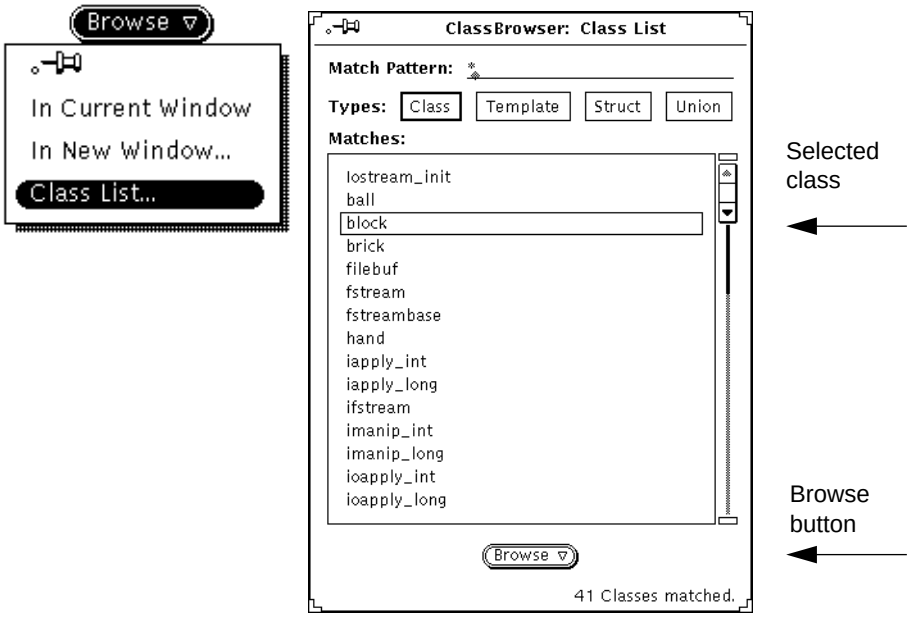
SourceBrowser activates the class in the ClassBrowser window. In this example, the user is browsing the class `block`.



If you are unsure of the classes in your program, you can call up an empty ClassBrowser window, then list all available classes to choose from.

1. **Select ClassBrowse from the Display window.**
   SourceBrowser displays an empty ClassBrowser window.

2. **Choose Class List from the Browse menu.**
   ClassBrowser activates the Class List window with an alphabetical list of all available classes in your program.



3. **Select a name in the Class List window, then click on the Browse button.**
   ClassBrowser displays the class in the current base window. ClassBrowser dismisses the List window if it is unpinned.

The following sections describe the contents of the ClassBrowser window after you issue a query.

Text Field

The class that ClassBrowser is actively browsing is called the *current class.* The current class name is displayed in the Class text field.

Base, Derived, and Friend Class Menus

The control area contains separate menus of the base, derived, and friend classes of the current class. The base and derived menus include the classes one level away from the current class. If a class has no base, derived, or friend classes, then the corresponding abbreviated menu button is inactive.

Data Members Display Pane

The Data Members display pane shows all data members defined in the current class. The data members are grouped by access protection: private, protected, and public. The names of inherited members are displayed, if requested.

Function Members Display Pane

The Function Members display pane shows all function members defined in the current class. The function members are grouped by access protection: private, protected, and public. The inherited members and friend functions are displayed, if requested.

The function members display pane may contain markers that identify the virtual and inline functions in the current class:

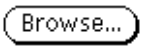
 Indicates a virtual function

 Indicates an inline function

 Indicates a virtual inline function

6.1.2 Activating ClassBrowser From ClassGrapher

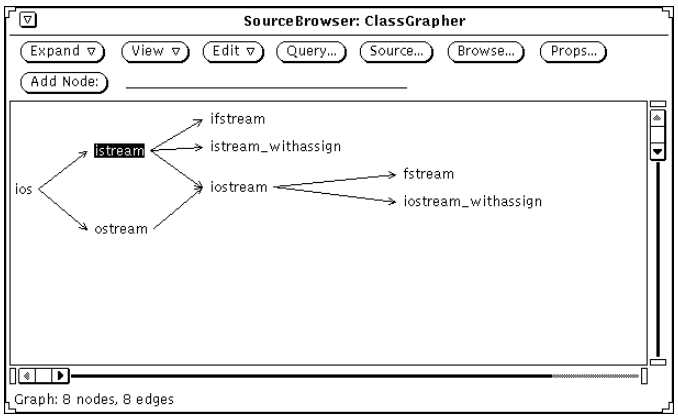
To activate ClassBrowser from the ClassGrapher window:



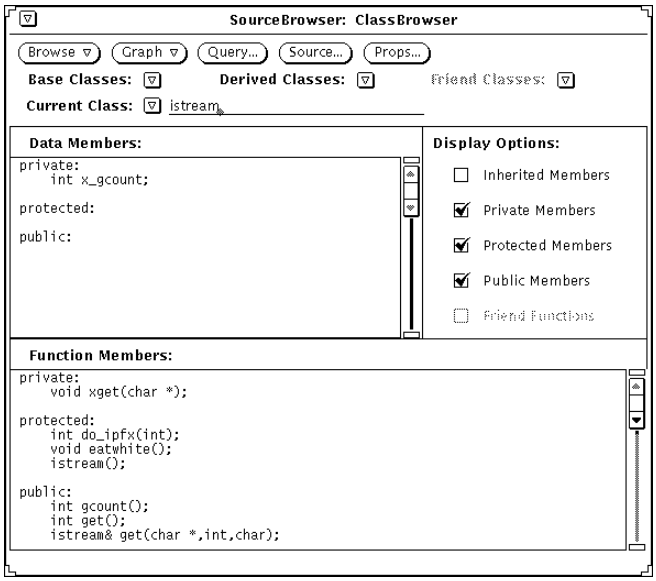
- **In the ClassGrapher, select the node you want to browse, then click on the Browse button.**
  If you select more than one node, then ClassBrowser browses the last selected class node.

In this example, the user selected the node `istream`.

Node selected in ClassGrapher



Current class in ClassBrowser is istream

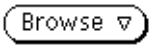


6.2 Browsing Subsequent Classes

You can browse additional classes in either the existing ClassBrowser window or in a new ClassBrowser window.

6.2.1 In the Existing ClassBrowser Window

To browse a class symbol in the existing ClassBrowser window:



- **Select the symbol you want to browse, then click on the Browse button.**
  You can select the symbol anywhere on the screen. This action is equivalent to choosing In Current Window from the Browse menu.

ClassBrowser offers four additional ways to browse a class in the existing window:

- Type the class name in the Text field and press Return.
- Choose a class name from the Base Classes, Derived Classes, or Friend Classes menu. See “Traversing the Class Hierarchy” on page 104 for more information.
- Choose a class from the class history menu.
- Select a class name in the Class List window, then click on the Browse button.

6.2.2 In a New ClassBrowser Window

SourceBrowser supports multiple ClassBrowsers. This feature is especially useful for exploring different classes at the same time. For example, you could display the class `brick` in one ClassBrowser window, and the class `block` in another to compare the two classes.

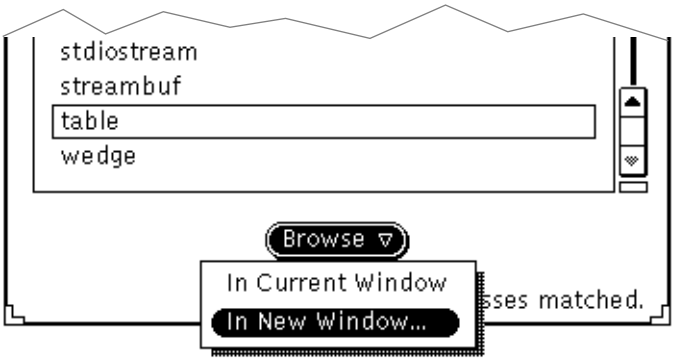
Multiple ClassBrowsers share the same Class List window. They do not share properties or class history.

To browse a class in a new ClassBrowser window, such that you have more than one ClassBrowser on the screen:

- **Select the symbol you want to browse, then choose In New Window from the Browse menu.**

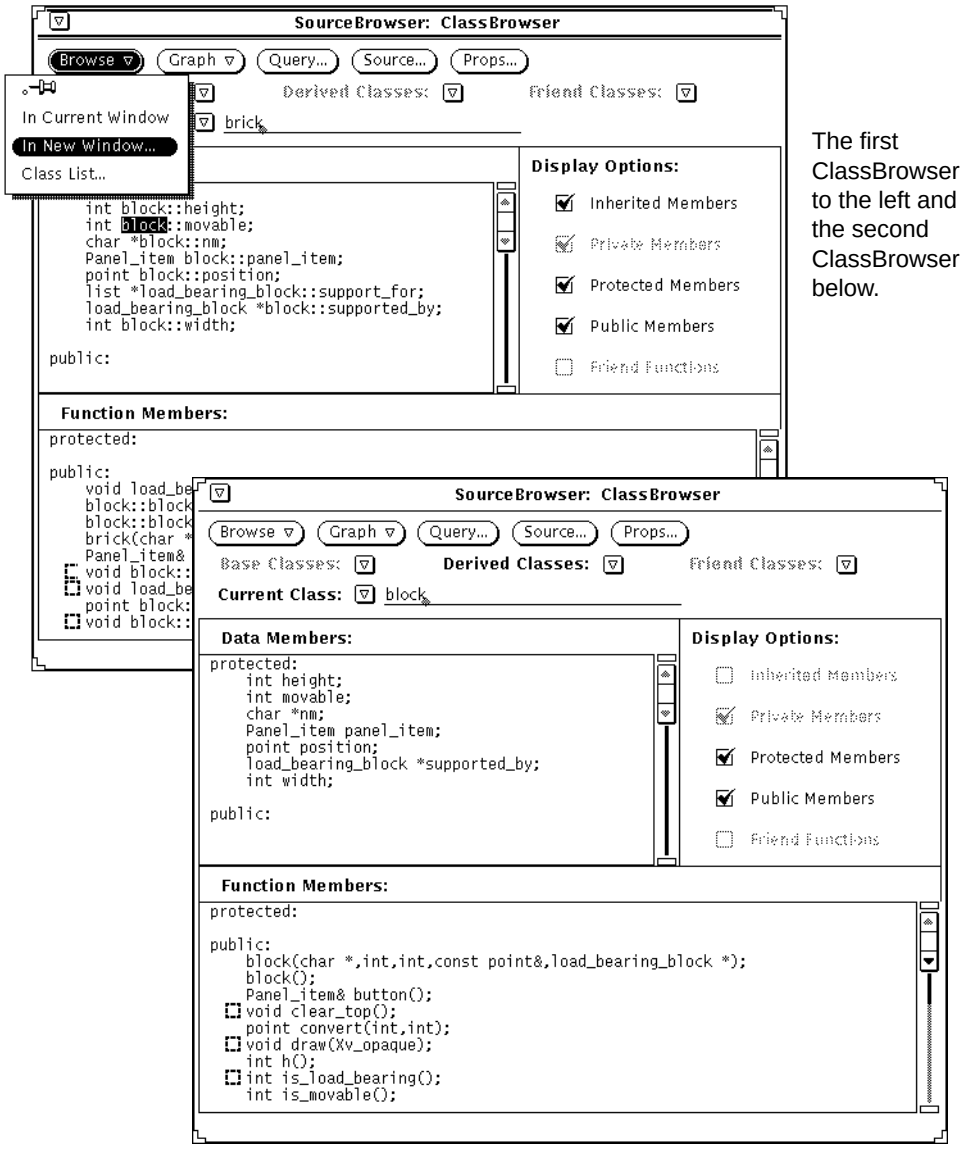
You can also activate multiple ClassBrowsers from the Class List window:

- **Select a name in the Class List, then choose In New Window from the Browse menu.**



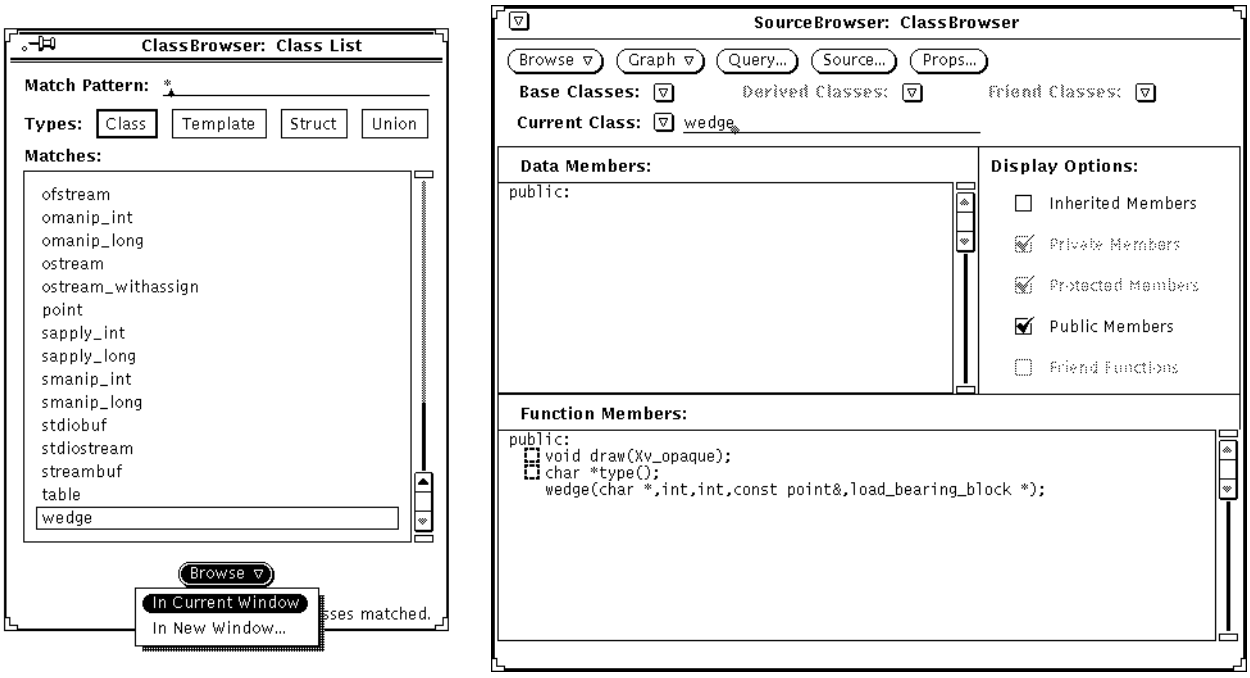
You can only activate multiple ClassBrowsers from ClassBrowser itself. When you browse a class from SourceBrowser or ClassGrapher, and multiple ClassBrowsers are running, then the class is displayed in the last ClassBrowser activated.

For example, suppose the user activated ClassBrowser from SourceBrowser or ClassGrapher with `brick` as the current class. The user turned on Inherited Members in the Display Options pane, selected the class `block` in the data members pane, then chose In New Window from the Browse menu. ClassBrowser activated a second window with `block` as the current class.



The first ClassBrowser is to the left and the second ClassBrowser is below.

Next the user activated the Class List window by choosing Class List from the Browse menu in the second ClassBrowser. The user selected wedge in the Class List window, then chose In Current window from the Browse menu. The class wedge became the current class in the second ClassBrowser.

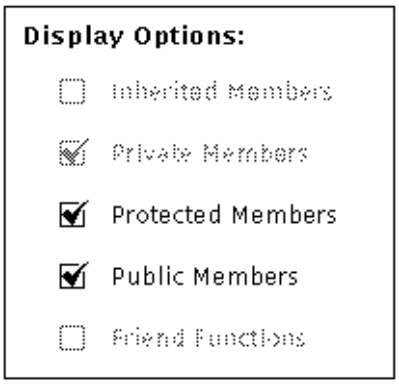


6.3 *Viewing Data and Function Members*

In the data and function member display pane, you can choose to display:

- Inherited members
- Private members
- Protected members
- Public members
- Friend functions

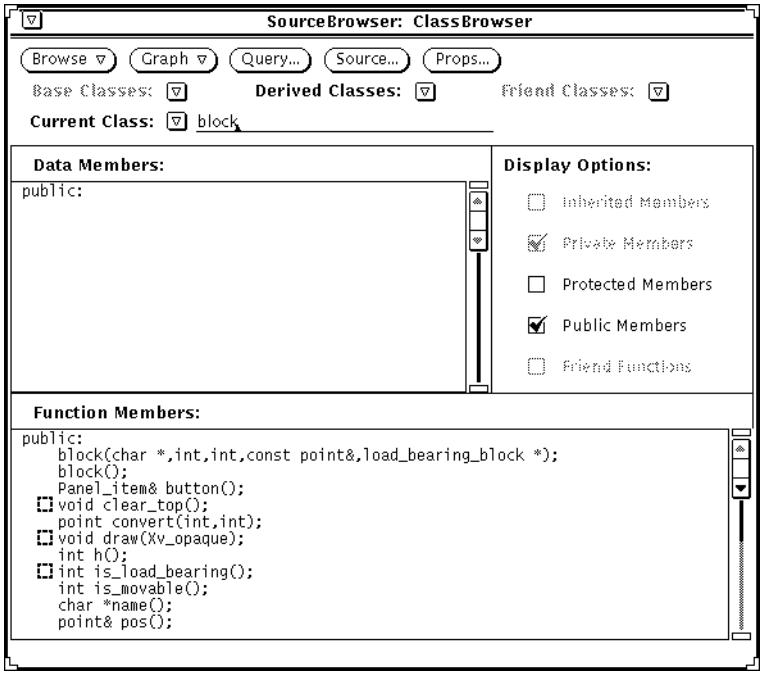
To change the display options:



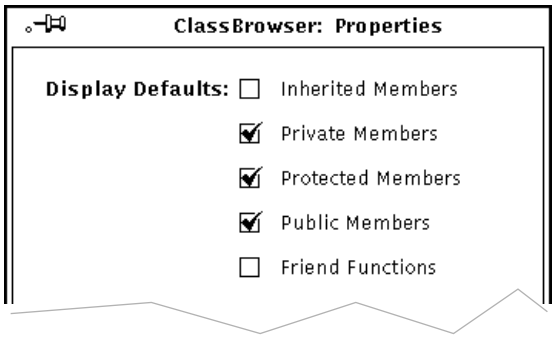
♦ **Click on the boxes in the Display Options pane.**

The display options are nonexclusive. If an option is inactive, then it does not apply to the current class. By default, ClassBrowser displays all private, protected, and public members of the current class.

In this example, the user turned off the display of protected members defined in the class `block`.



You can also turn on and off the display options in the Display Defaults field in the ClassBrowser Properties window.



Use the Properties window when you want to retain the changes in the `.SBdefaults` files in your home directory. Subsequent ClassBrowser sessions use these default property settings. The options in the Properties window are never inactive.

The changes you make in the Properties window are applied immediately to the ClassBrowser base window. The changes you make in the base window do not affect the Properties window.

6.4 *Viewing Virtual and Inline Functions*

You can identify the virtual and inline functions in the current class. ClassBrowser uses the following glyphs:

- Indicates a virtual function.
- Indicates an inline function.
- Indicates a virtual inline function.

To change the display:

1. **Click on the Props button**
2. **Click on a Show Glyphs option in the Properties window.**

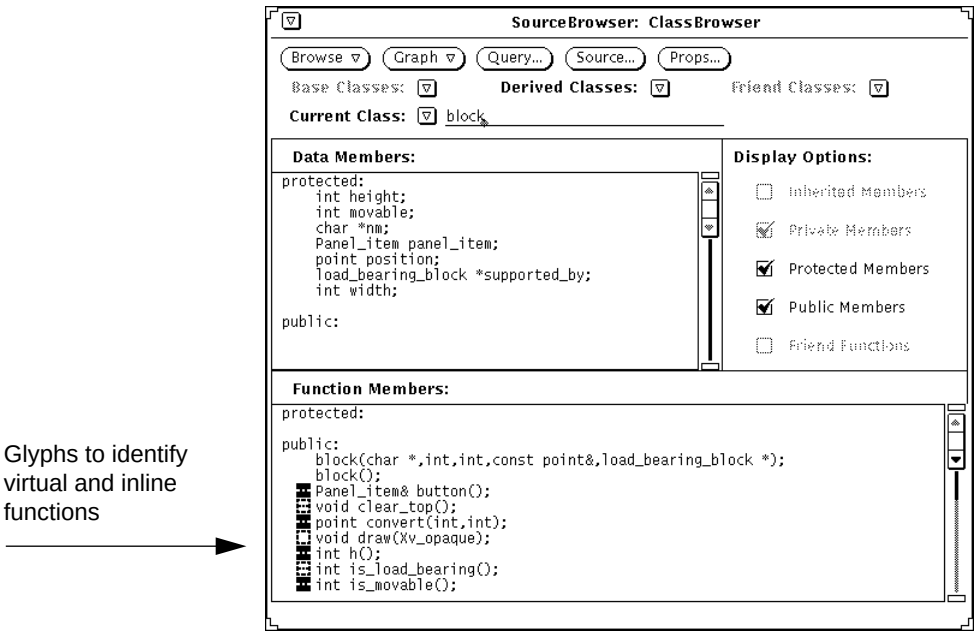
   The nonexclusive options are:

   Virtual—Identify virtual functions

Inline—Identify inline functions

3. Click on the Apply button.

In this example, the user turned on the display of both virtual and inline functions.



6.5 Traversing the Class Hierarchy

To browse a base, derived, or friend class of the current class:

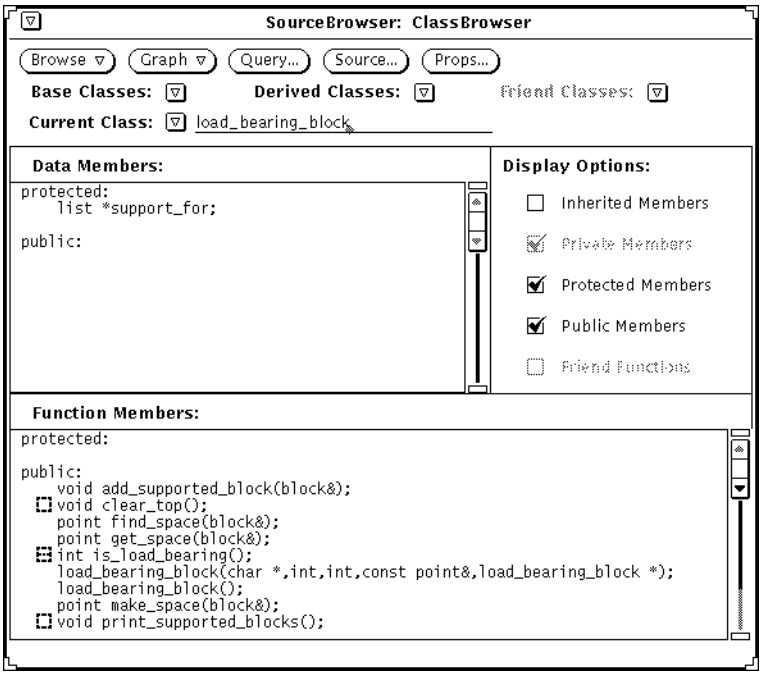
- **Choose a class from the Base Classes, Derived Classes, or Friend Classes menu.**

The selected class becomes the current class. The data and function members panes show the information relative to the new current class. The base, derived, and friend menus are updated accordingly.

If you click on the abbreviated menu button, then ClassBrowser displays the class that is the first menu item. If the current class has no base, derived, or friend classes, the menu button is grayed out accordingly.

In this example, the user selected `load_bearing_block`, which is a derived class of `block`.

Base, derived, and friend class menus

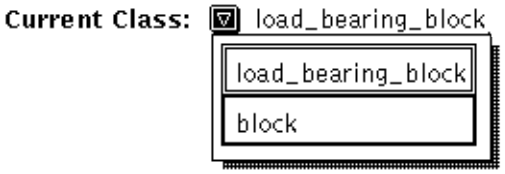


6.6 *Revisiting Previously Browsed Classes*

ClassBrowser maintains a history of the classes you have browsed during the current session in the Class menu. ClassBrowser always enters the new class as the first item in the menu.

To revisit a previously browsed class:

♦ **Choose a class from the current Class menu.**



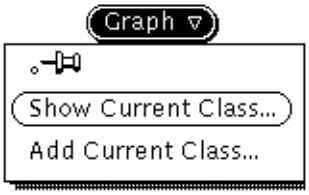
6.7 Graphing Class Inheritance

You can graph the hierarchy of the current class in the ClassGrapher window. You can graph the hierarchy either manually or automatically.

6.7.1 Manually Graphing a Class

To graph the class hierarchy manually:

♦ **Choose an item from the Graph menu.**



Show Current Class—SourceBrowser clears the ClassGrapher window before it graphs the current class.

Add Current Class—SourceBrowser adds the current class to the existing graph.

The graph includes *n* level of base and derived classes, where *n* is the expansion depth set in the ClassGrapher Properties window. It does not include friend classes.

6.7.2 Automatically Graphing a Class

To graph a class automatically, when it becomes the currently browsed class:

♦ **Open the ClassBrowser Properties window and set Graph mode to one of the Auto settings. Then click on the Apply button.**

The Auto settings are:

Auto: Show Current Class—Bring up the ClassGrapher window with the graph of the current class only.

Auto: Add Current Class—Add the current class to the existing graph.

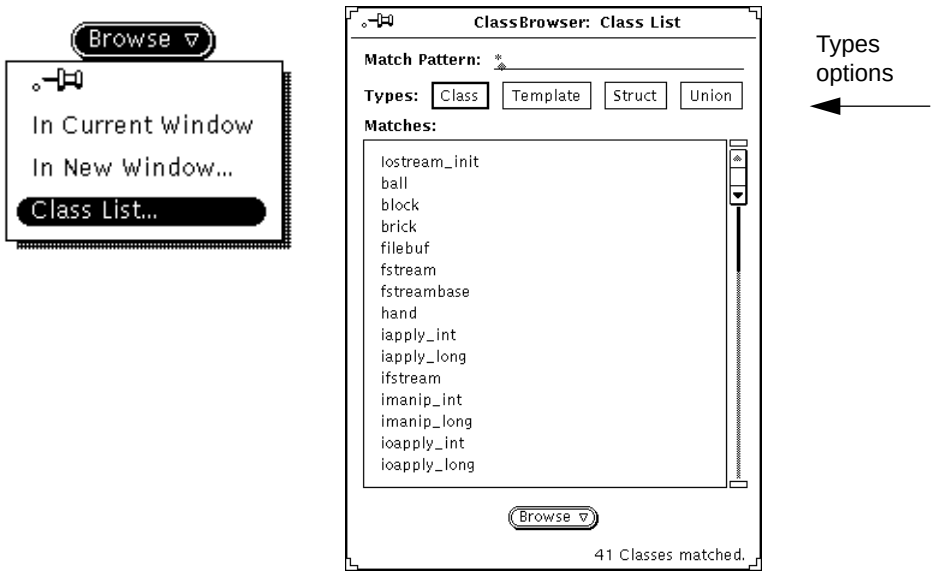
The default setting is Manual; to graph the class, you must go through the Graph menu in the ClassBrowser base window.

6.8 Listing Class Types

In the Class List window, you can list all `classes`, `templates`, `structs` and `unions` in your program.

To display the class list:

1. Choose Class List from the Browse menu.



2. Click on a Types option.

The nonexclusive options are:

Class—A class is a type that is user-definable. A class contains:

- A set of data members of various types
- A set of member functions for manipulating the class object

- A set of restrictions on the access to these data and function members

For example:

```
class block {
    protected:
        char *nm;
        int movable;
        .....
    public:
        block();
        char *name() {return nm:}
        int is_movable() {return movable;}
        .....
};
```

Struct—A struct is a class declared with the class-key `struct`; its members and base classes are public by default. For example:

```
struct bar {
    bar(int, char *);
    void set_width (int);
    .....
};
```

ClassBrowser puts only C++-style structs in the list. The general C-style `struct` (for example, `struct bar {int count; char *name;};`) is not shown.

Union—A union is a class declared with the class-key `union`; it can contain objects of different types at different times. For example:

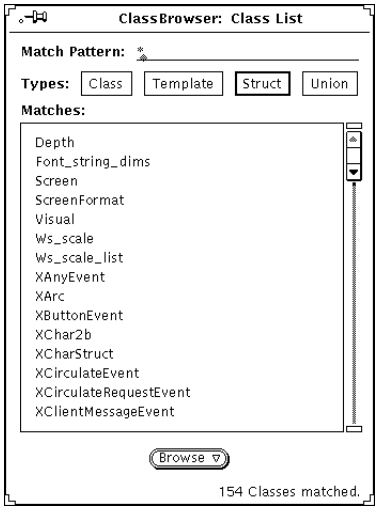
```
union tok_val {
    char *p;
    long i;
    double d;
        tok_val(const char *);
        tok_val(int ii) {i = ii;}
        tok_val(double dd) {d = dd;}
};
```

ClassBrowser lists only C++-style unions in the class list. The general C-style `union` (for example, `union star {char *s; int i;};`) is not shown.

Template—A template is an instantiated template class. For example:

```
template<class T> class Vector {
    T* v;
    int sz;
public:
    Vector(int);
    T& operator[] (int);
    T& elem(int i) {return v[i];}
};
```

In this example, the user displayed all structs in the Blocks program.

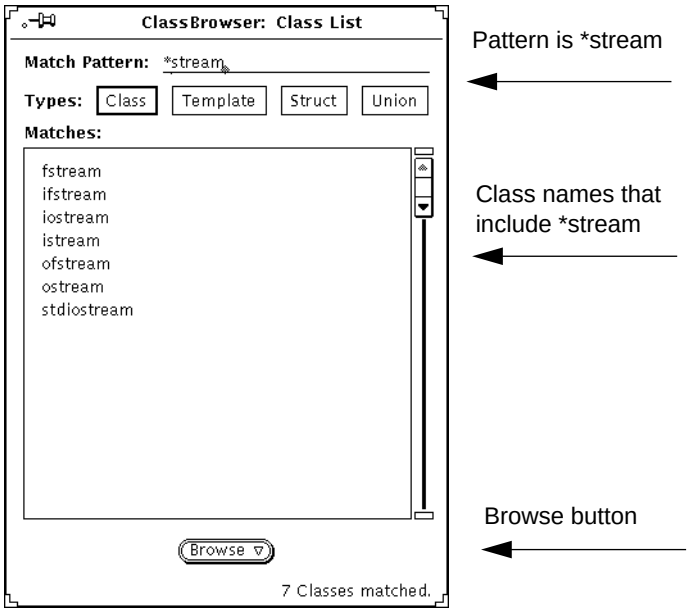


List of structs in Blocks program

To list a subset of classes:

- **Type the match pattern that represents the subset in the Text Field and press Return.**

In this example, the user found all classes that matched the pattern `*stream`.



To browse a class in the list:

- **Select a name, then click on the Browse button.**
  If multiple ClassBrowsers are running, then the class is displayed in the window from which the Class List was last activated.

6.9 Querying for Classes and Functions

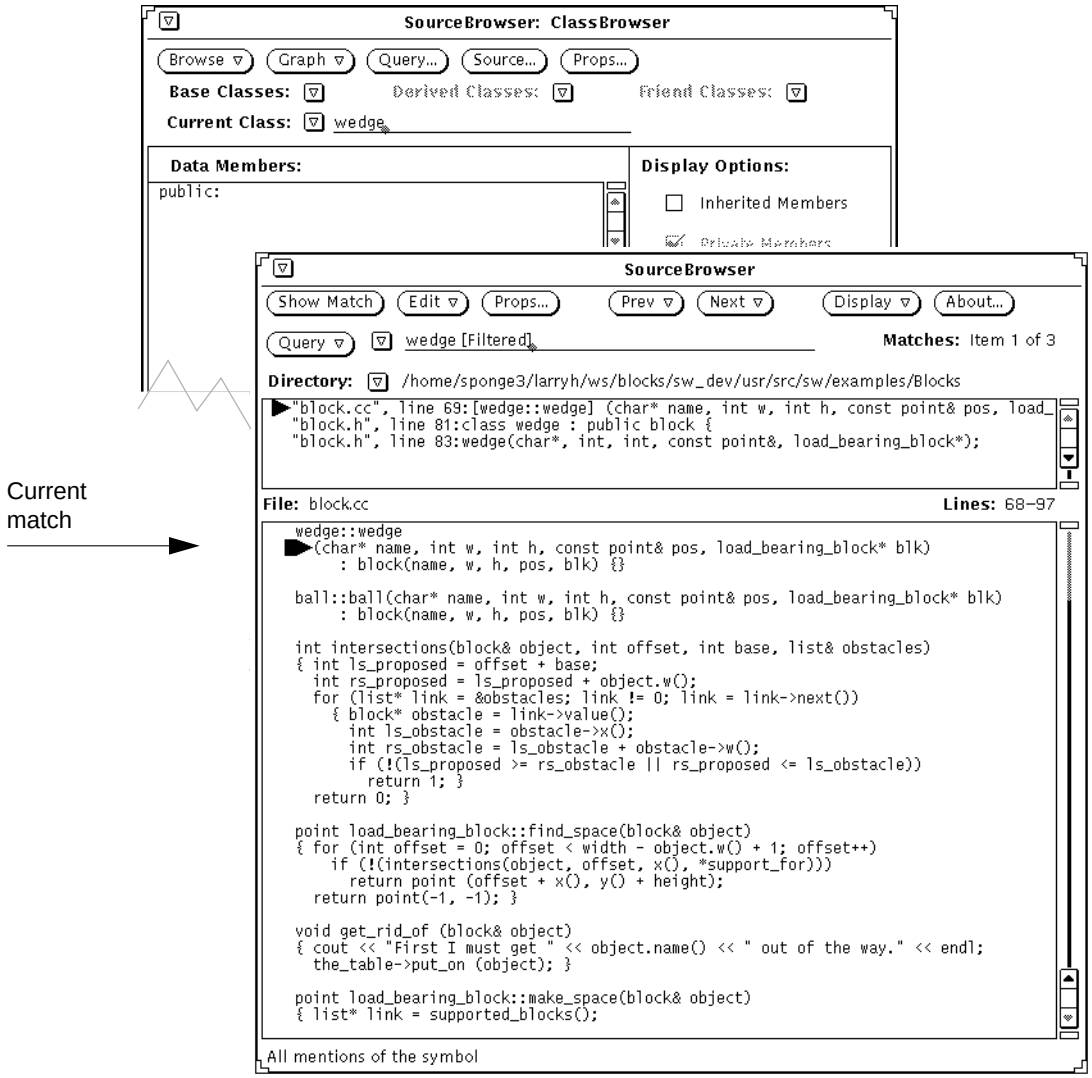
You can issue a query on a class, data member, member function, or friend function from the ClassBrowser window. ClassBrowser displays the results of the query in the SourceBrowser base window.

To issue a query from the ClassBrowser window:

- Select the symbol then click on the Query button. If you do not select a symbol, then ClassBrowser issues a query on the current class.

Caution – The symbol must be selected from a text pane, not a grapher node.

The Query function is especially useful for finding out where a function being declared is referred. In this example, the user queried on the class wedge.



6.10 Displaying the Source of Classes and Functions

You can display the source code of any class, member function, or friend function from the ClassBrowser window.

Note – You cannot ask for the source code of the variable name from data members. If you do so, you will receive an error message.

You can display the source in either the SourceBrowser or the Debugger window.

6.10.1 Displaying Source in SourceBrowser

To display the source of a symbol in the SourceBrowser base window from the ClassBrowser:

1. **Set Show Source In to Browser in the ClassBrowser Properties window.** This setting is the default.
2. **Select a symbol, then click on the Source button in the ClassBrowser window.**

- If you do not select a symbol, then ClassBrowser displays the source of the current class.
- If you select an overloaded function, then ClassBrowser finds the right variant of the function in the source code.

In this example, the user displayed the source code for a constructor for the class `block` in the SourceBrowser window. Note that two `block` constructor functions are defined in the source and that ClassBrowser displayed the source for the correct variant.

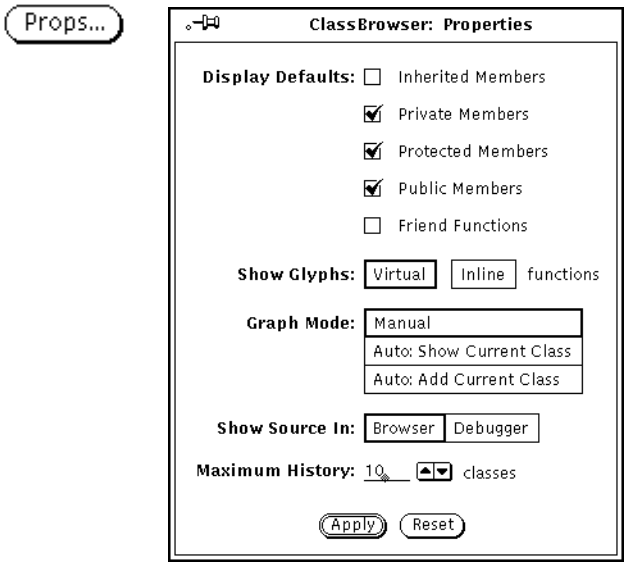
The beginning of the source is marked with a hatched arrow.

6.11 Customizing ClassBrowser

Use the Properties window to customize the properties associated with the ClassBrowser.

To open the ClassBrowser Properties window:

- **Click on the Props button.**



To set a property:

- **Enter the desired value or click on the desired setting, then click on the Apply button.**
  ClassBrowser dismisses the window if it is unpinned. Any changes you make to the Properties window are retained in the `.SBdefaults` file in your home directory. All subsequent SourceBrowser sessions use the default properties settings in the `.SBdefaults` file.

To discard any changes you have made in the Properties window, but have not yet applied:

- **Click on the Reset button.**
  ClassBrowser returns the properties to their last applied values. The Properties window remains open.

Descriptions of the ClassBrowser properties follow.

Display Defaults

The check boxes control what ClassBrowser displays in the Data Member and Member Functions panes. To change your display, click on the box next to the desired option. You can turn on and off the display of inherited, private, protected, and public members and friend functions. The display options are nonexclusive.

The changes you make in the Properties window are immediately applied to the base window when you click on the Apply button. If you want to change the display options for the current class only, then use the Display Options settings in the ClassBrowser base window.

The private, protected, and public members are on by default.

Show Glyphs

Show Glyphs tells ClassBrowser whether or not to mark virtual and inline functions in the member functions display pane.

Virtual—Mark virtual functions with a hollow box. This setting is on by default.

Inline—Mark inline functions with a solid box with a line in it.

The Virtual and Inline settings are nonexclusive. When both settings are on, ClassBrowser marks virtual inline functions with a hollow box with a line through it.

Graph Mode

Graph Mode tells ClassBrowser whether or not to automatically graph a class in the ClassGrapher window when it becomes the currently browsed class.

Manual—Use the Graph menu in the ClassBrowser base window to graph the current class. This setting is the default.

Auto: Show Current Class—Automatically graph the current class and its immediate base and derived classes in the ClassGrapher window. The ClassGrapher clears the window before graphing the class.

Auto: Add Current Class—Automatically add the current class and its immediate derived and base classes to the existing graph in the ClassGrapher window.

Show Source In

Show Source In tells ClassBrowser where to display the source code of a class when you click on the Source button.

Browser—Display the source of the class in the SourceBrowser source pane.

Debugger—Display the source of the class in the Debugger window.

Maximum History

The History menu records the most recently browsed classes, including the currently browsed class. Click the up or down arrow buttons to the desired value. The minimum history is 1, and the maximum history is 30. The default is 10.

6.12 Quitting ClassBrowser

Note – It is also possible to close the ClassBrowser to an icon. Once you dismiss the window, you cannot retrieve the class information. To browse the class again, you need to activate a new window from either the SourceBrowser or ClassGrapher; or a remaining ClassBrowser, if you have multiple ClassBrowsers running.

To quit ClassBrowser:

♦ **Choose Quit from the ClassBrowser header menu.**

Restricting the Match Set 7

This chapter describes how to use the SourceBrowser Filter and Focus commands to limit your search for symbols. With the Filter command, you can search for symbols based on the way in which they are used in a program. The Focus command limits your search to certain kinds of code, such as files or functions.

This chapter is organized into the following sections:

Filtering a Query	*page 119*
Focusing A Query	*page 129*

7.1 Filtering a Query

You can use filtering to:

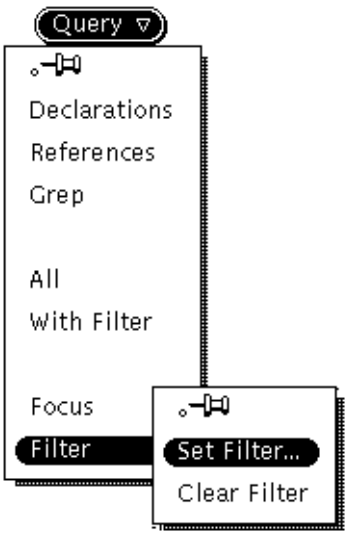
- Obtain matches for very specific uses of a symbol
- Reduce the number of matches you receive

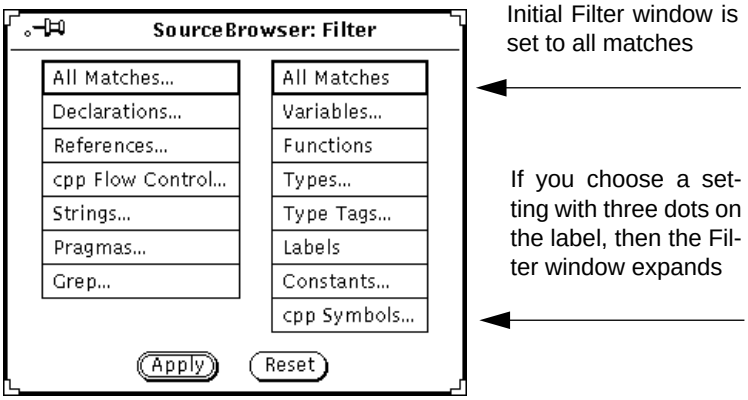
See the section “-help_filter” on page 218 for information on the equivalent command-line option.

7.1.1 Setting the Filter

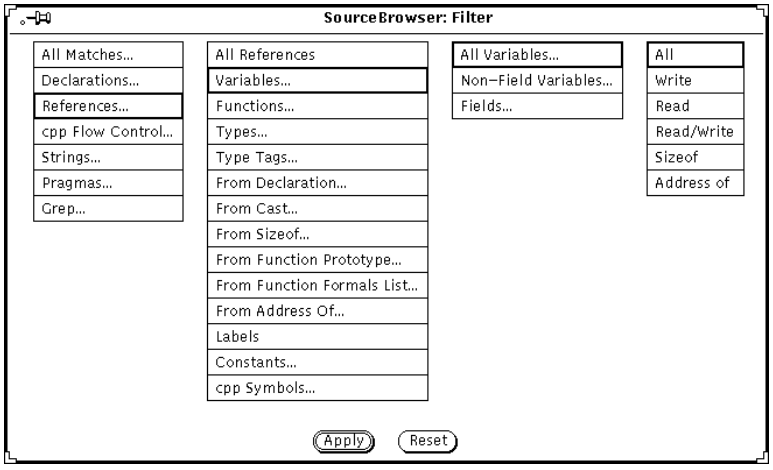
To set the filter:

1. **Choose the Filter ➤ Set Filter item in the Query menu.**
   SourceBrowser displays the Filter popup window. The content of the Filter window is determined by the language(s) of the source code you are browsing.





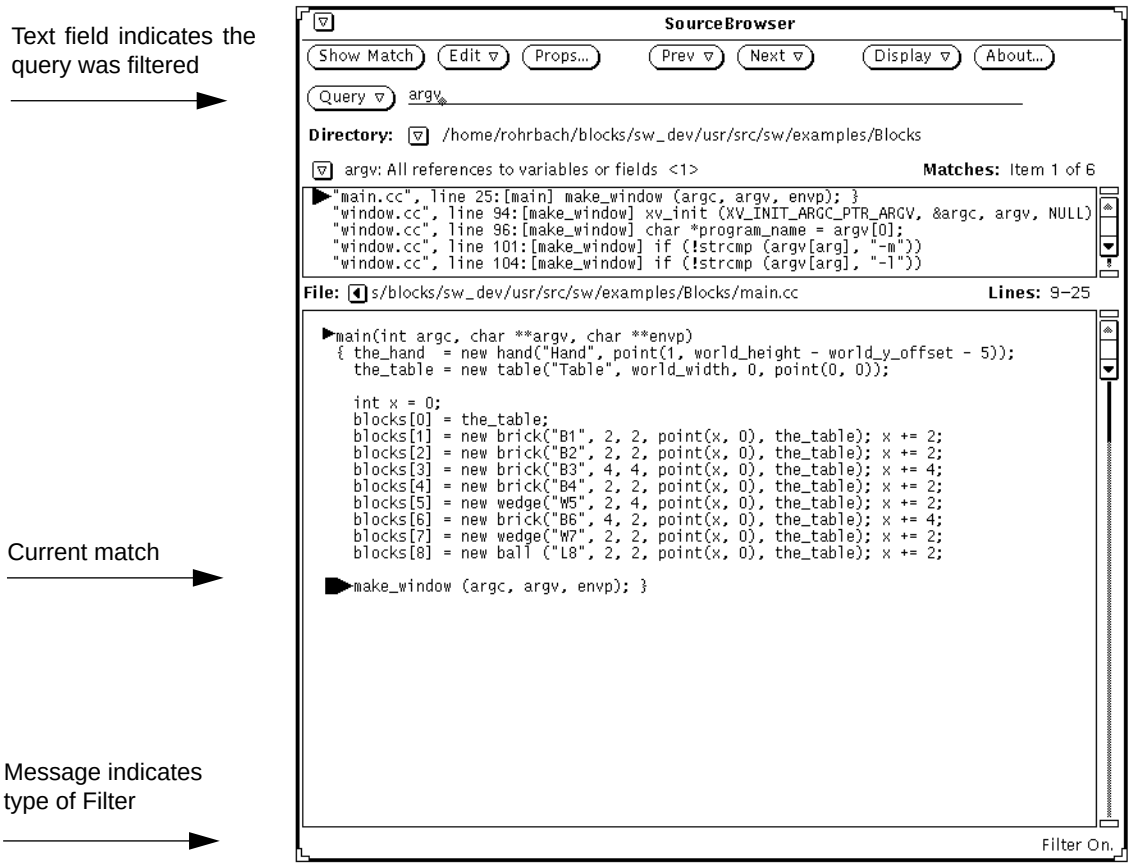
2. **Click on the setting that describes the filter to use in your search.**
   If the setting label has three dots (...), then the Filter window expands to include additional settings, which you can use to further narrow your search. This filter tells SourceBrowser to find only those places in the source code where a variable is modified.



3. **Click on the Apply button.**
   The right footer in the SourceBrowser base window displays a message indicating that filtering is turned on. The left footer indicates the type of Filter you set.

   This filter will be used on all subsequent queries.

Now issue a query. In this example, SourceBrowser found six places where `argv` is modified.



Once the filter has been set, it will be applied to all subsequent queries. To return to doing unfiltered queries, you must clear the filter.

The following is important information about setting a filter:

- You can only have one Filter setting for any query.
- When you choose a different current query, the settings in the Filter window change to reflect the settings you used when you issued that query. If you are moving between queries with different filters, then you can avoid confusion by leaving the Filter window in view.
- You can issue a filtered query for a symbol that already has an active query. SourceBrowser labels the filtered query in the Query menu.
- If SourceBrowser does not accept the Filter you have set, then check the Parse Query String setting in your Properties window. If either From Selection or From Text Field is turned on, then SourceBrowser ignores the Filter.

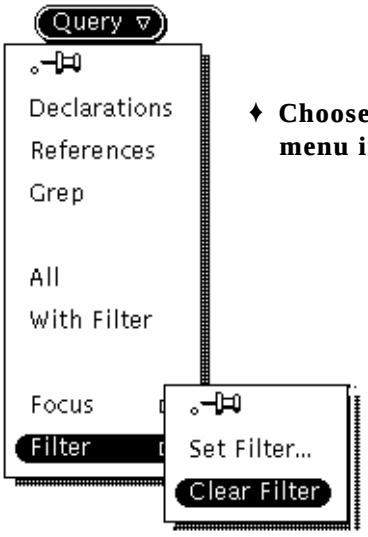
7.1.2 Resetting the Filter

To return the filter display window to the last filter you applied:

♦ **Click on the Reset button in the Filter window.**
SourceBrowser discards the changes you have made, but have not yet applied in the Filter window. The Filter window remains open so you can make additional changes.

7.1.3 Clearing the Filter

To reset the Filter:

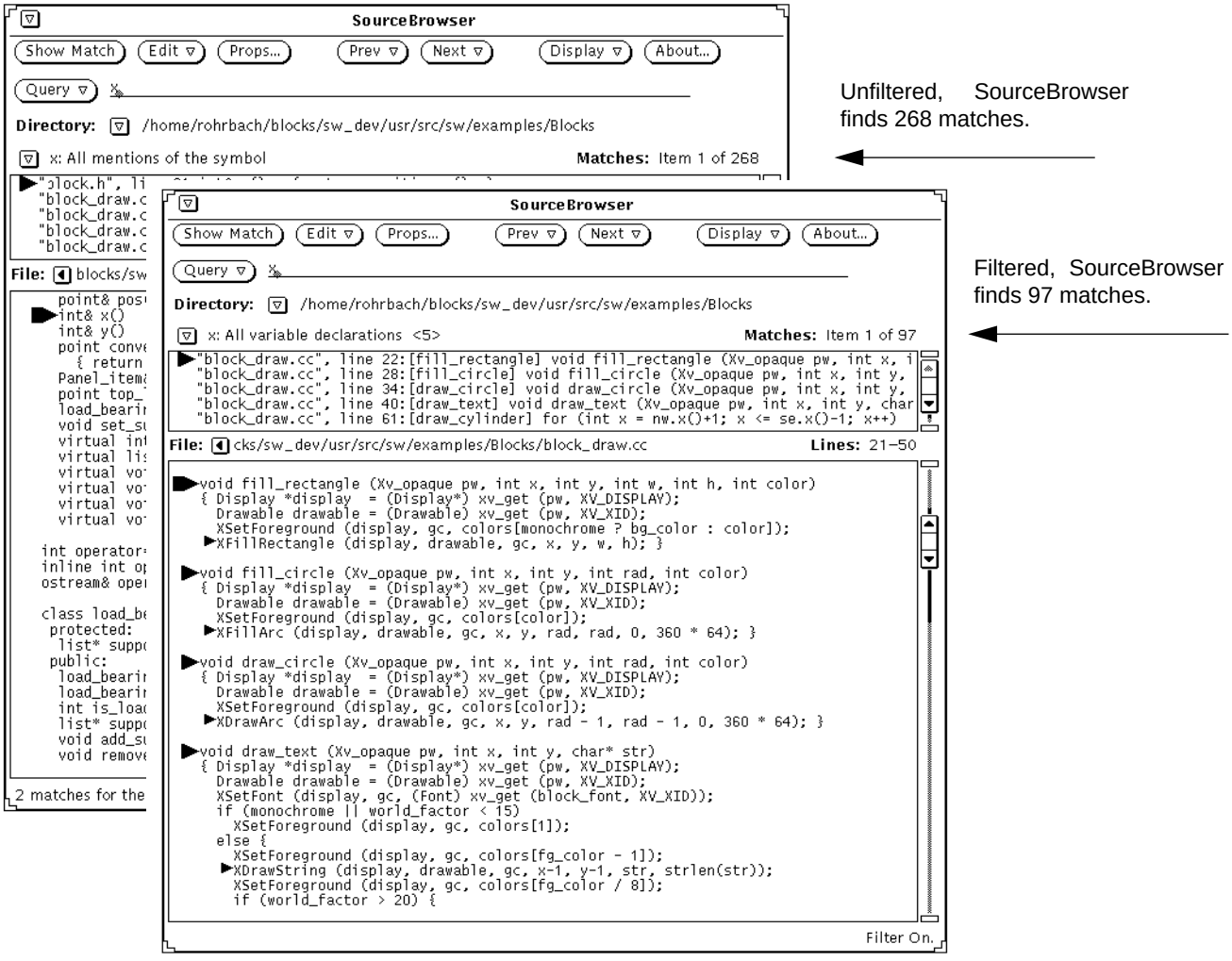


♦ **Choose the Filter ➤ Clear Filter item from the Query menu in the SourceBrowser base window.**

7.1.4 Filtering to Reduce the Number of Matches

You can use the Filter command to reduce the number of matches you receive. For example, when you query on the symbol x in the Blocks program, SourceBrowser returns 268 matches. In this case, SourceBrowser produces such a large number of matches that the information present is not very useful.

If you are only interested in seeing which instances of x are used as declarations, you can set the Filter to declarations of variables, then reissue the query. In this case, SourceBrowser returns 97 matches.



Notice that the Match line indicates SourceBrowser found 66 secondary matches. See the next section for an explanation.

7.1.5 Filtering in Macro Definitions

When SourceBrowser browses through source code with macros, it is unable to determine how identifiers inside macro definitions are used. For example, in the following code fragment, SourceBrowser does not know how `times` in the macro definition will be used because it depends on the context in which SECONDARY_DEMO is used.

```
#define SECONDARY_DEMO times

...

    int   SECONDARY_DEMO = 1;

...

    SECONDARY_DEMO++;
```

In this case, SourceBrowser is not sure whether to show a match, so it shows a match on the definition of the macro (but not on the usages of the macro), just in case. This match is known as a *secondary match*. A further search on the macro name can be performed to find out where `times` may be used via the macro. SourceBrowser marks secondary matches with a black arrow with a question mark in the source pane:

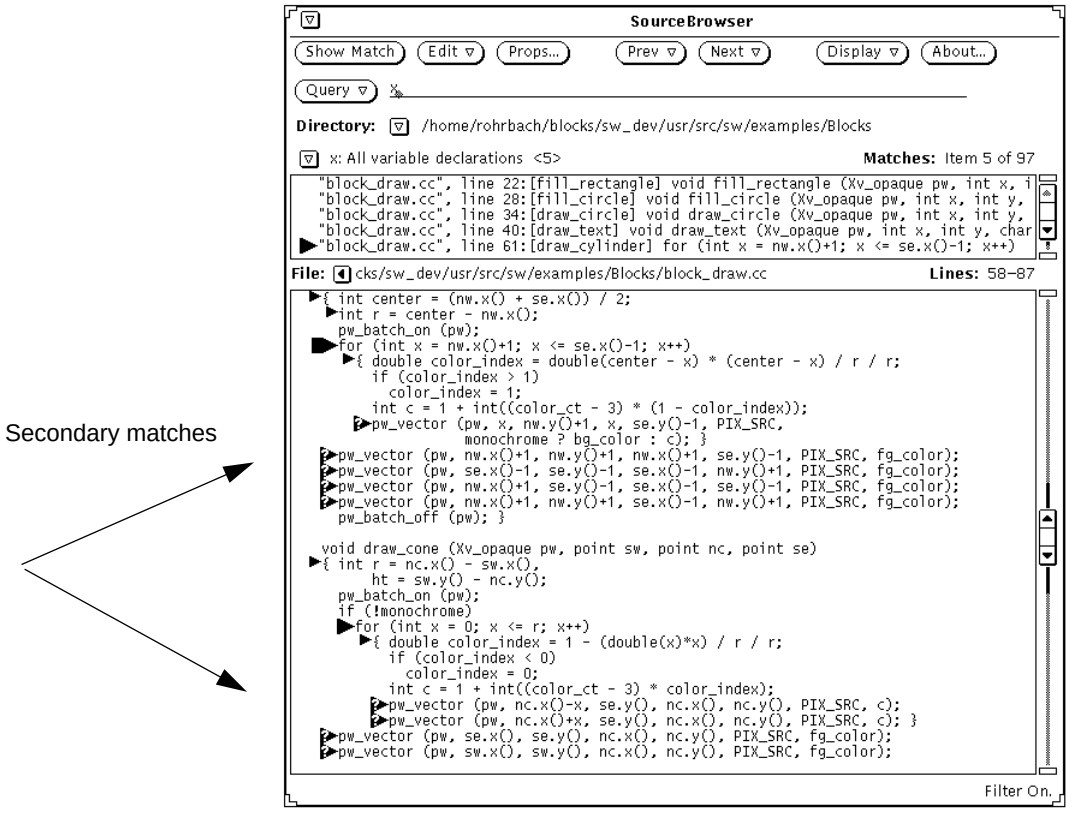

Indicates a secondary current match


Indicates a secondary match other than the current match found by the current query


Indicates a secondary match found a query other than the current query

SourceBrowser returns secondary matches *only* when you query with a filter. SourceBrowser also displays symbols given as actual parameters of macro expansions as secondary matches.

In this example, the user used the Next command to view the secondary matches found by the current query.



If secondary matches are not of interest to you, then you can turn off their display:

- **Set Return Secondary Matches to No in the Properties window.**

This action saves memory because SourceBrowser now discards all secondary matches at query time.

If you decide later you want to see secondary matches, then you must reset Return Secondary Matches to Yes and reissue your queries.

7.1.6 Filtering Different Uses of the Same Symbol

Sometimes the same symbol may be used in several different ways in a program. For example, *next* may be both a variable and a file in a structure. You can use filtering to separate these matches.

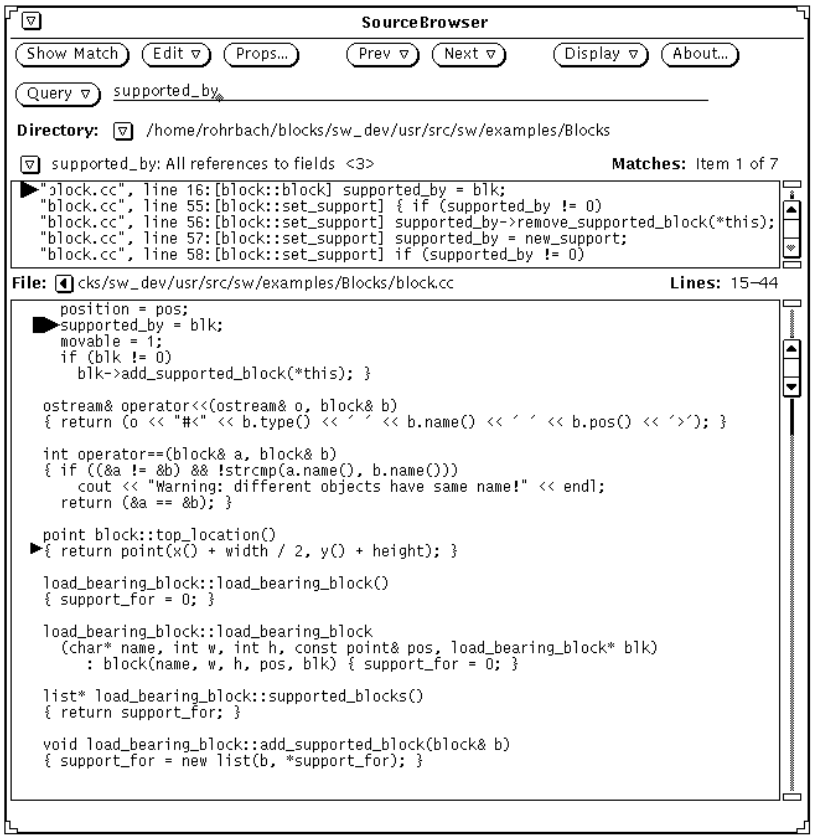
7.1.7 Filtering to Obtain Very Specific Matches

You can also use the Filter command to issue a very specific query such as "Show all occurrences of the variable `age` when `age` is written to as a structure field name."

For example, suppose the user queried for the field `supported_by` in the Blocks program. When filtering is turned off, SourceBrowser returns eight matches, including the field and all of its uses.

When the user set a filter to show all occurrences in which a field is assigned a value and then queried for `supported_by`, SourceBrowser returned two matches.

The first match is in the constructor for `block`, where `supported_by` is first assigned a value. The second match is in the `block` member function `set_support`. The user can now be confident that `set_support` maintains the `supported_by` field.



SourceBrowser finds seven matches of `supported_by`

Another typical use of Specific Matches is to distinguish between reads and writes.

7.2 *Focusing A Query*

You can restrict SourceBrowser to query only certain sections of the source: for example, all source files written in a specific language or a particular set of functions. This restriction is known as focusing.

Use focusing when you want to:

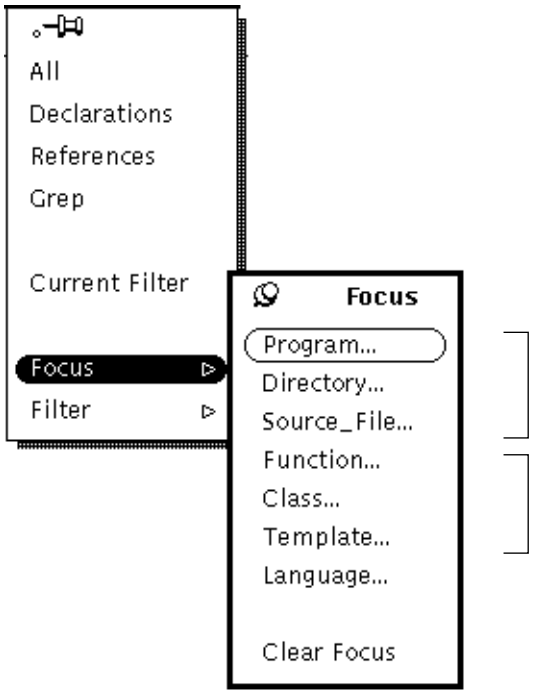
- View only a subset of your source
- Reduce the number of matches you receive

See the section “-help_focus” on page 217 for information on the equivalent command-line option.

7.2.1 Setting the Focus

To set the focus:

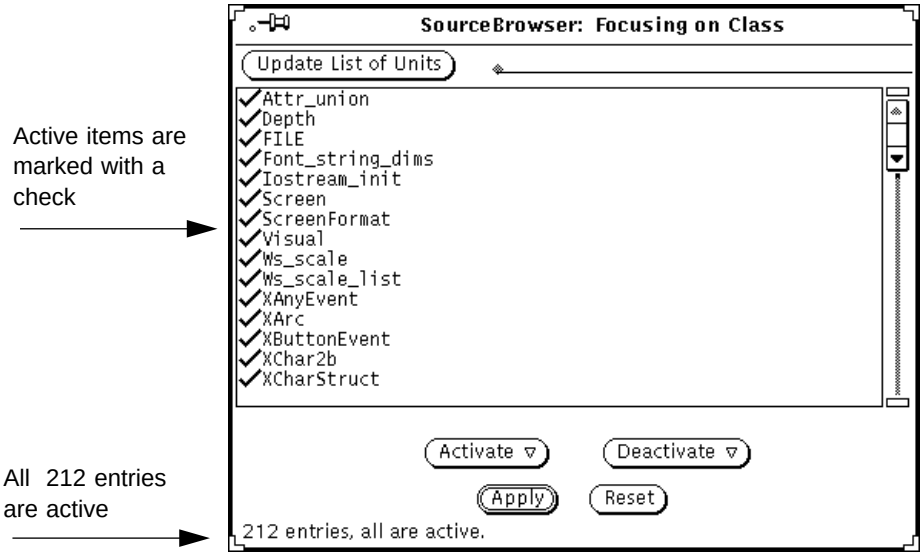
1. **Choose the type of unit you want to focus on from the Focus submenu on the Query menu.**
   SourceBrowser displays the Focus popup window with all of the available items for the selected unit.



If you choose any of these types or the language type, then you will focus on an entire file(s).

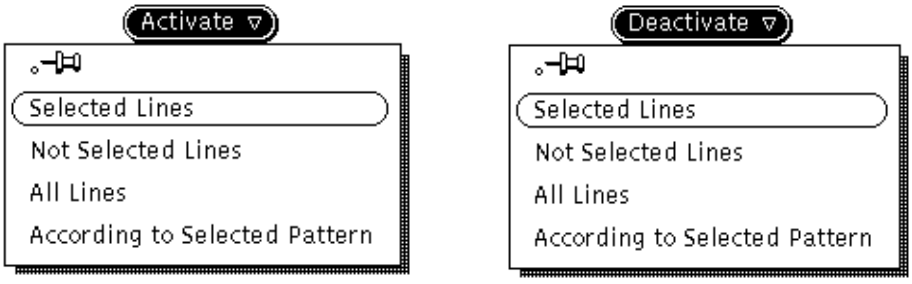
If you choose Function, Macro, Class, or Template, then you will focus on a piece of code, not an entire file.

In this example, the user chose Class from the Focus submenu. This window



shows a list of all classes in the database. By default, all classes are in the focus (activated).

2. Include or Exclude the items you want to use in your query by means of the Activate and Deactivate menus.

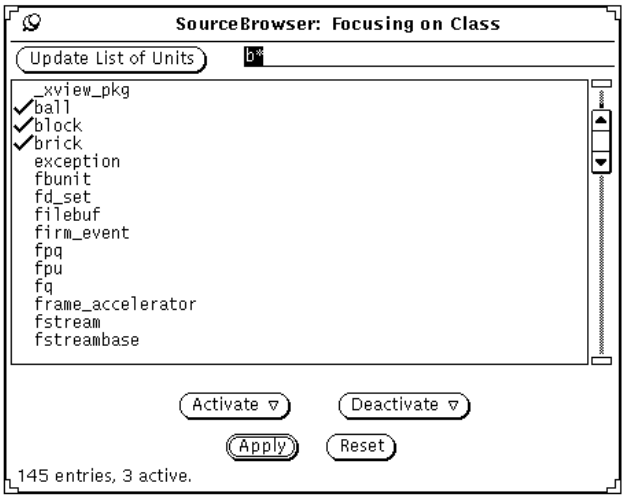


The menu items are:

- **Selected lines**—Activate / deactivate selected lines.
- **Not Selected Lines**—Activate / deactivate all lines that are not selected.

- **All Lines**—Activate / deactivate all lines.
- **According to Selected Pattern**—Activate / deactivate items based on the selection pattern you have typed in on the text field at the top of the window . The selection pattern may contain shell-style or regular expression wildcards, depending on the Properties setting.
- In the example below, the user limited the search to identifiers found in classes that begin with the letter “b” by first choosing All Lines from the Deactivate menu. The user next typed and selected `b*` in the Text Field, then chose According to Selected Pattern from the Activate menu. SourceBrowser found three instances of the pattern using shell-style pattern wildcards.

Active items are checked

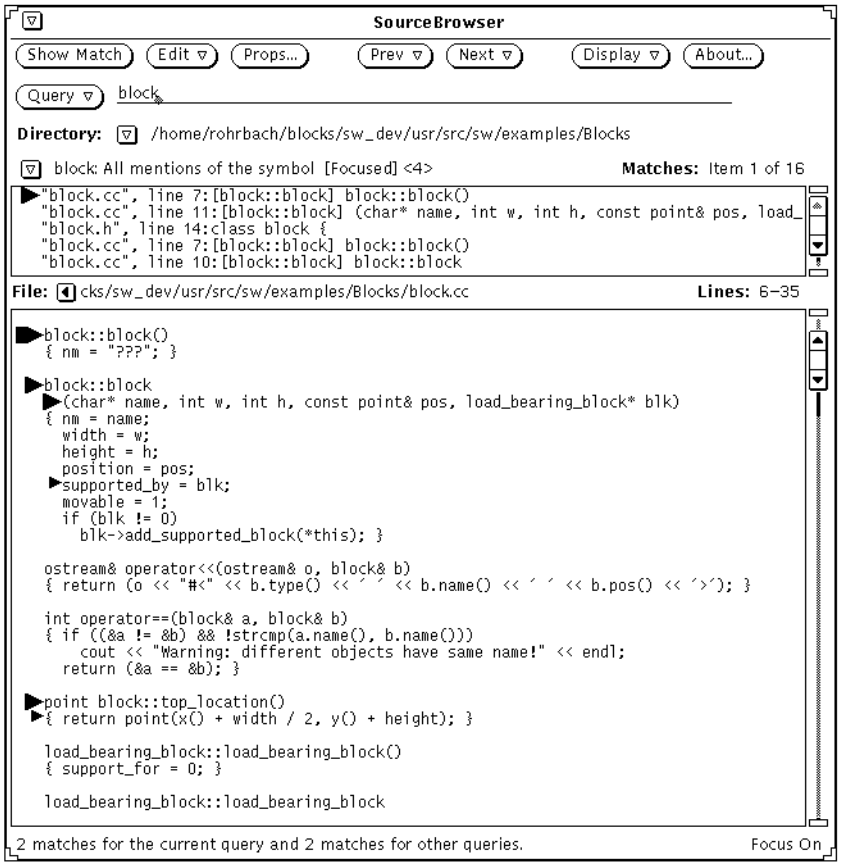


3. **Click on the Apply button.**
   The footer in the SourceBrowser base window displays a message indicating that focusing is turned on (see Page 128).

An asterisk appears on the Focus unit in the Focus menu to show that a selection is active.

Now issue a query. In this example, the user searched for all occurrences of block on the classes block, brick, and ball. SourceBrowser returned 5 matches.

Indicates the query was focused



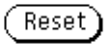
The following is important additional information about setting a focus:

- You can select units on more than one focus selection. Each of the active selection are marked with an asterisk in the Focus menu. In the example above, if you also selected function main, all queries would be focused on the source code for classes block, brick, and ball together with the source code for main.
- Once you use the Focus command to specify a focus, the focus remains in effect for subsequent queries until you either apply different settings in the Focus window or clear the focus.

- If you get the error message `No focus units of that type are available`, then SourceBrowser cannot find anything to focus on. For example, you might get this message if you try to focus on classes and your C++ program has no classes.
- If SourceBrowser does not accept the Focus you set, then check the Parse Query String setting in your Properties window. If either From Selection or From Text Field is turned on, then SourceBrowser ignores the Focus. Instead, SourceBrowser expects you to do your focusing using command-line options.

7.2.2 Resetting the Focus Window

To return to the last focus selection you applied:



♦ **Click on the Reset button in the Focus window.**
SourceBrowser discards the changes you have made, but have not yet applied, in the Focus window. The Focus window remains open so you can make additional changes.

7.2.3 Updating the Focus Window

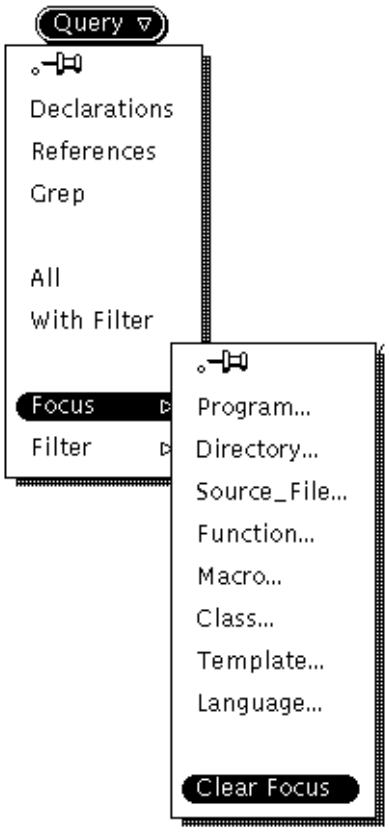
If you have edited and compiled your source code so that the unit of code you are focusing on has changed, then you will want to update the Focus window with the new information.

To update the contents of the Focus window:

♦ **Click on the Update List of Units button in the Focus window.**

7.2.4 Clearing the Focus

To clear the Focus:



♦ **Choose Clear Focus from the Focus menu. (All focus selections are cleared and the asterisks marking them are deleted.)**

7.2.5 Focusing on Selected Files

By default the SourceBrowser searches all source files that have been included in the database. A single database may contain files from multiple programs in multiple directories. This can be useful, so that you do not need to change databases when you are browsing one program, then want to browse a different program. Conversely, if the code from multiple programs and directories are contained in one database, a single query may return matches

from more than one program. For example, every C program has a function named *main*. If you query on *main*, the SourceBrowser will find a match for the *main* function of each program in the database.

Focusing allows you to narrow down the query, eliminiating spurious matches. For example, you can focus on a single program or on source code files located in a single directory or set of directories, or on a single source file or set of source files. In addition, you can focus on the code for certain types of program object such as functions and macros, and for C++, classes and templates. For example, if the variable *P* is defined locally in many different functions, you can focus on a single function to see matches for only one local definition of *P*.

7.2.6 Focusing on More than One Unit of Code

You can use the Focus command to focus your query on more than one unit of code. For example, you could conduct a query that was limited to certain source files. Or, you might want to focus on one class and one file. Your query would catch the definition of the class and its member functions and the uses of the class in the selected file.

Integrating Editing and Debugging 8

This chapter describes how to integrate editing and debugging with the SourceBrowser. This chapter is organized into the following sections::

Editing Source Code	*page 139*
Using SourceBrowser with the Debugger	*page 141*
Using an External Editor with the SourceBrowser and Debugger	*page 150*

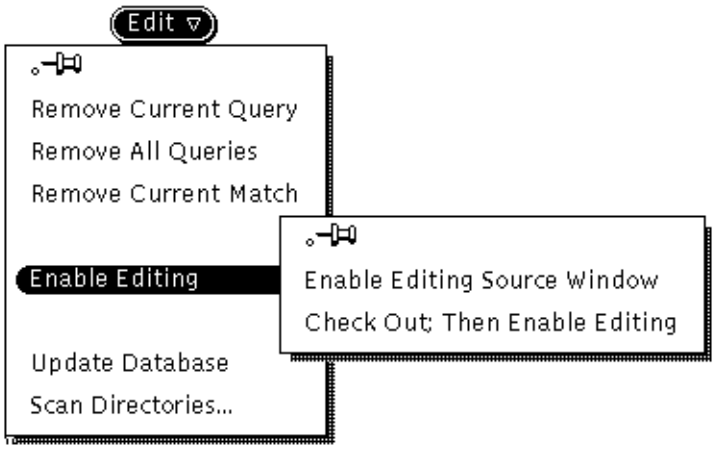
The first section describes how to edit source code from within the SourceBrowser using the OpenWindows Text Edit command. The second section shows how to synchronize the SourceBrowser source display with the Debugger's source display. Both displays use OpenWindows' Text Edit display. The third section shows how to use a conforming external editor to replace both the SourceBrowser and the Debugger's source displays.

8.1 Editing Source Code

You can edit source code without leaving SourceBrowser. You use OpenWindows Text Edit commands.

To edit code:

1. **Choose Enable editing from the Edit menu.**



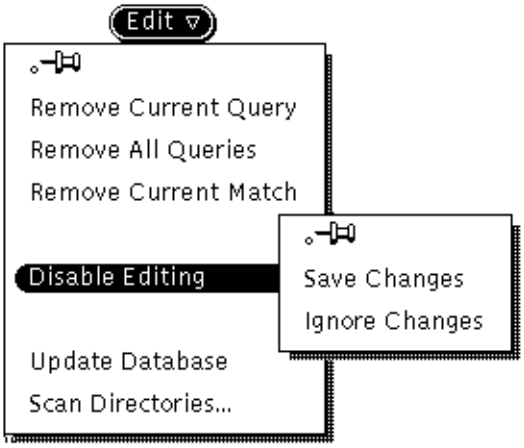
- If your code is not controlled by the Source Code Control System (SCCS), then choose Enable Editing Source Window from the Enable Editing submenu.
- If the code is controlled by SCCS, then choose Check Out; Then Enable Editing from the Enable Editing submenu. Do this step even if the file is already checked out.

Note – SourceBrowser removes the glyphs from the source pane so that they do not interfere with your editing.

---

2. **Edit the code in the source pane.**

3. Disable editing from the Edit menu.



- To save your changes, choose Save Changes from the Disable Editing submenu.
- If you do not want to save your edits, then choose Ignore Changes from the Disable Editing submenu.

If you edit the source file and then use either the Next or Prev command to move to a different file, SourceBrowser pops up a notice asking whether you want to save the edits before changing files.

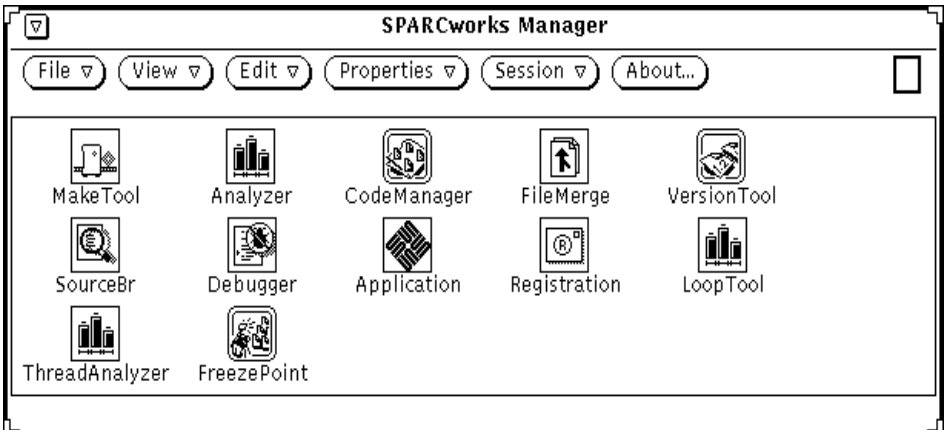
Note – You cannot check files back in to SCCS from SourceBrowser. You must check them in from a Shell window.

8.2 Using SourceBrowser with the Debugger

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

Before you can synchronize SourceBrowser with the Debugger, you must perform the following steps:

1. **Ensure that you started both the Debugger and SourceBrowser from the same Manager.**
   If not, quit the tools. Then, open a new Manager and activate the Debugger and SourceBrowser by double clicking on their icons. For details on using the Manager, see the manual, *Managing the Toolset*.



2. **Set SrcDispSync to Always in the Debugger Window (Use the Props, Category, Miscellaneous menus, in that order).**
   Otherwise, SourceBrowser cannot display source in the Debugger window. For details on using the Debugger, see the Sun manual, *Debugging a Program*.

3. **Ensure that the program you are browsing with SourceBrowser is properly loaded into the Debugger.**
   Otherwise, you cannot set a breakpoint in the Debugger. Use the Load item in the Debugger Program menu to load your program if necessary.

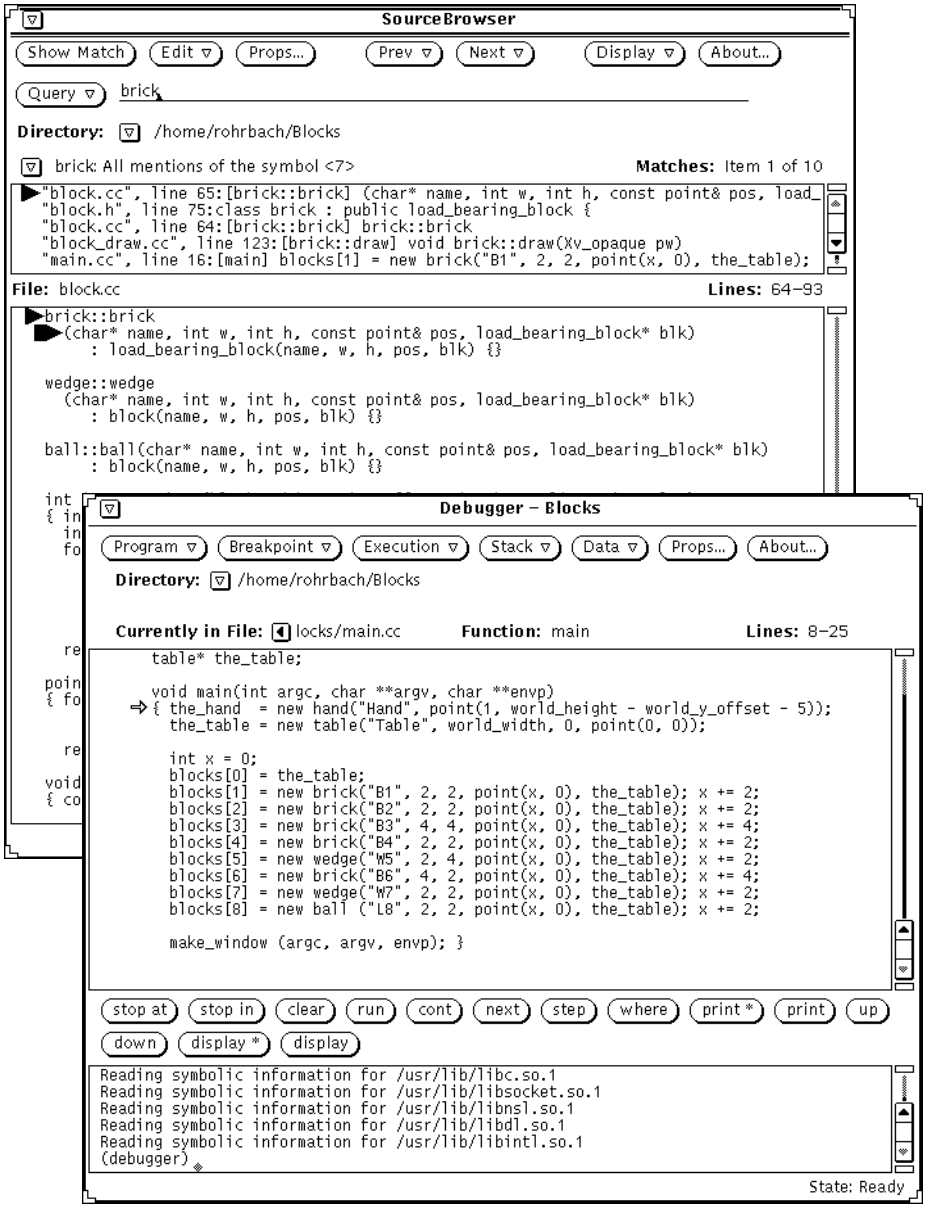
   To synchronize the Debugger with the SourceBrowser:

   Use the SourceBrowser to display the line of code where you want to place a breakpoint.

4. Choose Sync Debugger from the SourceBrowser Display menu.

The code that appeared in the SourceBrowser now also appears in the Debugger.

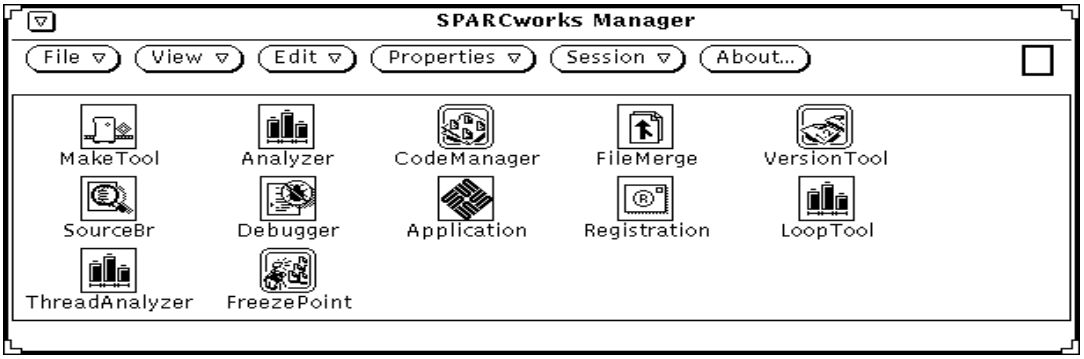
5. Use the Debugger to set the breakpoint in its own source display.


SourceBrowser
Show Match
Edit
Props...
Prev
Next
Display
About...
Query
brick
Directory: /home/rohrbach/Blocks
brick: All mentions of the symbol <7>
Matches: Item 1 of 10
"block.cc", line 65:[brick::brick] (char* name, int w, int h, const point& pos, load_
"block.h", line 75:class brick : public load_bearing_block {
"block.cc", line 64:[brick::brick] brick::brick
"block_draw.cc", line 123:[brick::draw] void brick::draw(Xv_opaque pw)
"main.cc", line 16:[main] blocks[1] = new brick("B1", 2, 2, point(x, 0), the_table);
File: block.cc
Lines: 64–93
brick::brick
(char* name, int w, int h, const point& pos, load_bearing_block* blk)
: load_bearing_block(name, w, h, pos, blk) {}
wedge::wedge
(char* name, int w, int h, const point& pos, load_bearing_block* blk)
: block(name, w, h, pos, blk) {}
ball::ball(char* name, int w, int h, const point& pos, load_bearing_block* blk)
: block(name, w, h, pos, blk) {}
Debugger – Blocks
Program
Breakpoint
Execution
Stack
Data
Props...
About...
Directory: /home/rohrbach/Blocks
Currently in File: locks/main.cc
Function: main
Lines: 8–25
table* the_table;
void main(int argc, char **argv, char **envp)
{ the_hand = new hand("Hand", point(1, world_height - world_y_offset - 5));
the_table = new table("Table", world_width, 0, point(0, 0));
int x = 0;
blocks[0] = the_table;
blocks[1] = new brick("B1", 2, 2, point(x, 0), the_table); x += 2;
blocks[2] = new brick("B2", 2, 2, point(x, 0), the_table); x += 2;
blocks[3] = new brick("B3", 4, 4, point(x, 0), the_table); x += 4;
blocks[4] = new brick("B4", 2, 2, point(x, 0), the_table); x += 2;
blocks[5] = new wedge("W5", 2, 4, point(x, 0), the_table); x += 2;
blocks[6] = new brick("B6", 4, 2, point(x, 0), the_table); x += 4;
blocks[7] = new wedge("W7", 2, 2, point(x, 0), the_table); x += 2;
blocks[8] = new ball ("L8", 2, 2, point(x, 0), the_table); x += 2;
make_window (argc, argv, envp); }
stop at
stop in
clear
run
cont
next
step
where
print *
print
up
down
display *
display
Reading symbolic information for /usr/lib/libc.so.1
Reading symbolic information for /usr/lib/libsocket.so.1
Reading symbolic information for /usr/lib/libnsl.so.1
Reading symbolic information for /usr/lib/libdl.so.1
Reading symbolic information for /usr/lib/libintl.so.1
(debugger)
State: Ready

8.2.1 Displaying Source for CallGrapher

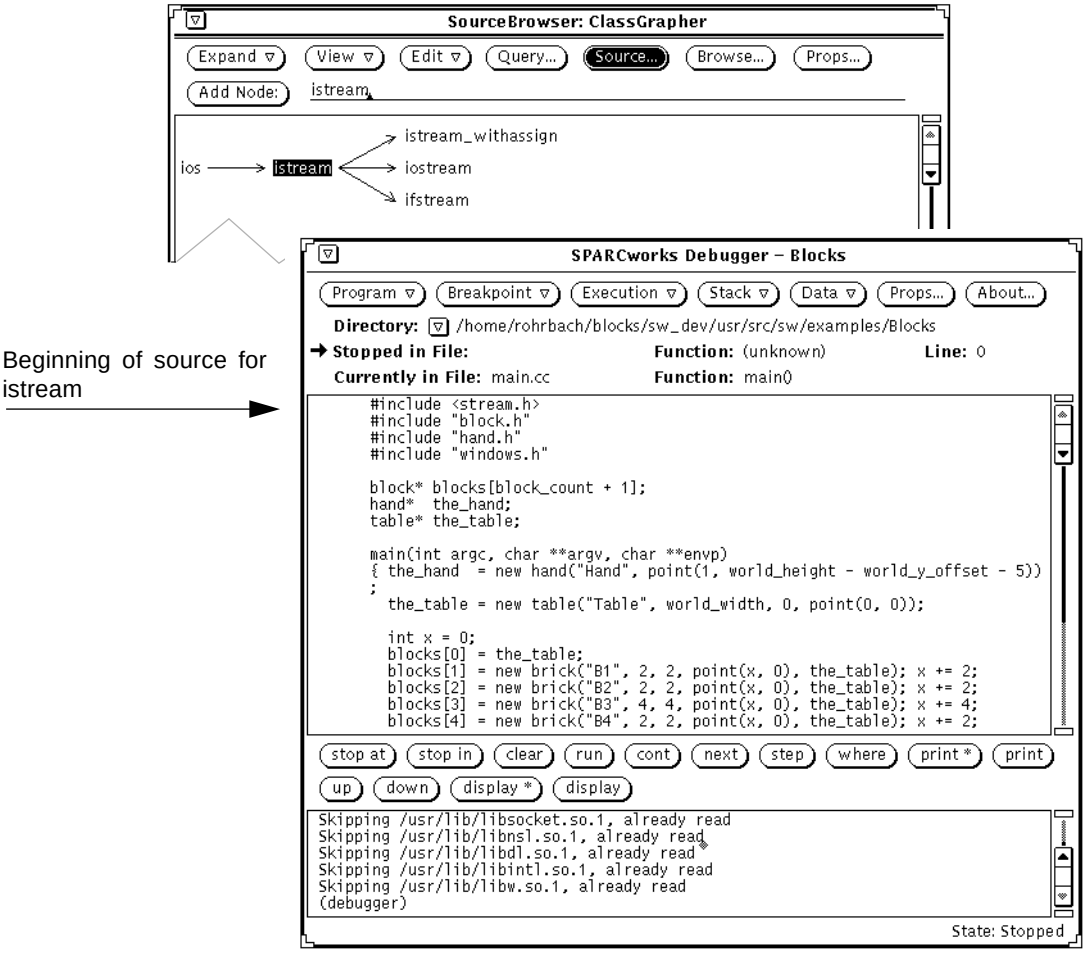
To display the source of a node in the Debugger:

1. **Ensure that you started both the Debugger and SourceBrowser from the same Manager.**
   If not, quit the tools. Then open a new Manager and activate the Debugger and SourceBrowser by double-clicking on their icons. For details on using the Manager, see the manual, *Managing the Toolset*.



2. **Ensure that the program you are browsing with SourceBrowser is properly loaded into the Debugger.**
   Otherwise, you cannot set a breakpoint in the Debugger. Use the Debug item in the Debugger Program menu to load your program if necessary.
3. **Set SrcDisplay Synchronization to Always in the Debugger Window. (Use the Props, Category, and WindowConfigs menus, in that order.)**
   Otherwise, SourceBrowser cannot display source in the Debugger window. For details on usingthe Debugger, see the Sun manual, *Debugging a Program*.
4. **Set "Show Source In" to Debugger in the CallGrapher Properties window.**
5. **Select the node you want to show the source of, then click on the Source button in the CallGrapher window.**

In this example, the user displayed the source for `istream`.



The following is important additional information about showing the source of a node in the Debugger:

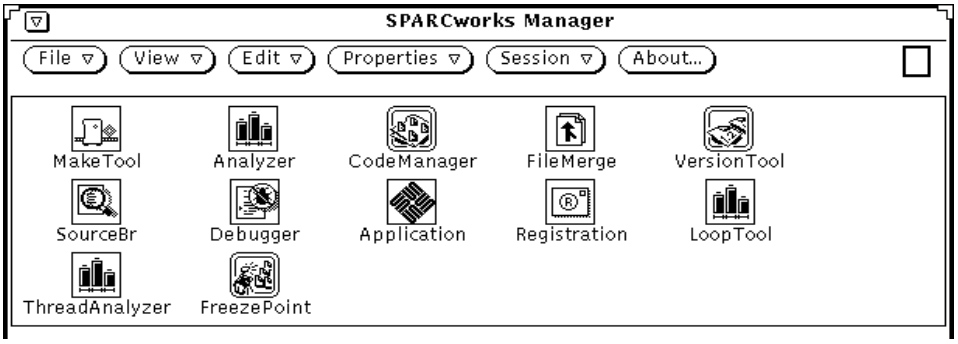
- If you receive the Debugger message `Request displaying` <*program name*> `denied`, then you have not set the Debugger SrcDisplays Synchronization property to Always. Set this property in the Debugger Window Configurations Properties window.

- If you try to set a breakpoint and you receive the message `No program`, then you have not properly loaded the program in the Debugger. Use the Debug item in the Program menu to load your program. For more information, see the Sun manual, *Debugging a Progra*m.
- If you are running more than one Debugger, then the Grapher displays the source in all the Debuggers that are:
  - Running in the same session as the Grapher
  - Have the SrcDisplays Synchronization property set to Always.
- If no Debugger is running when you click on the Source button, then nothing happens.
- You can run the Debugger in any directory.
- You can query on any symbol in the Debugger window by selecting the symbol, then pressing the SourceBrowser Query button.

8.2.2 Displaying Source for ClassBrowser

To display the source of a symbol in the Debugger:

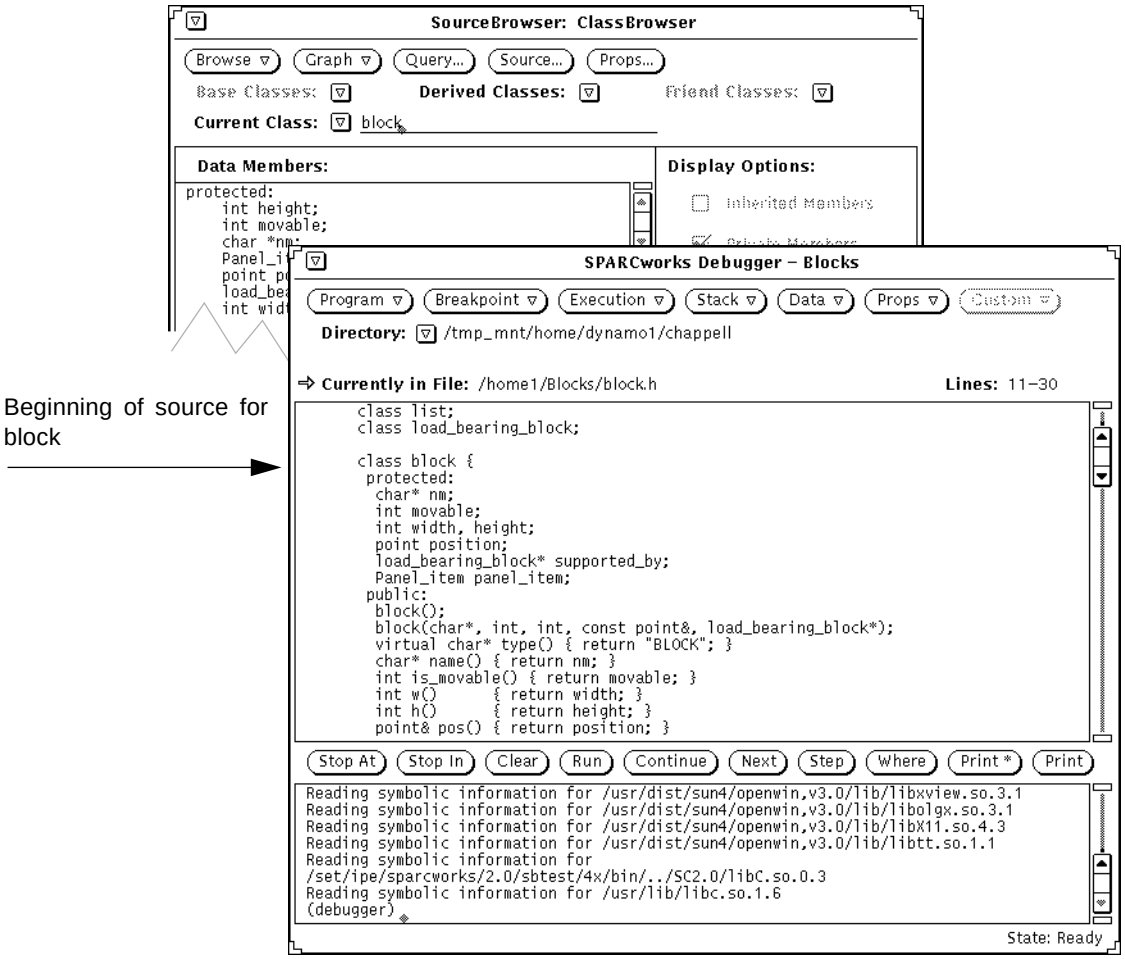
1. **Ensure that you started both the Debugger and SourceBrowser from the same Manager.**
   If not, then quit the tools. Then open a new Manager and activate the Debugger and SourceBrowser by double clicking on their icons. For details on using the Manager, see the manual, *Managing the Toolset*.



2. **Ensure that the program you are browsing with ClassBrowser is properly loaded into the Debugger.**
   Otherwise, you cannot set a breakpoint in the Debugger. Use the Debug item in the Debugger Program menu to load your program if necessary.

3. **Set SrcDisplay Synchronization to Always in the Debugger Window Configurations Properties window.**
   Otherwise, SourceBrowser cannot display source in the Debugger window. For details on using the Debugger, see the manual, *Debugging a Program*.
4. **Set Show Source In to Debugger in the ClassBrowser Properties window.**
5. **Select a symbol anywhere on the screen, then click on the Source button in the ClassBrowser window.**

In this example, the user displayed the source code for the class `block` in the Debugger window.



The following is important additional information about showing the source of a symbol in the Debugger:

- If you receive the Debugger message `Request displaying` <*program name*> `denied`, then you have not set the Debugger SrcDisplaySync property in the Debugger window to Always. Use Props, Category Miscellaneous, in that order.

- If you try to set a breakpoint and you receive the message `No program`, then you have not properly loaded the program in the Debugger. Use the Debug item in the Program menu to load your program. For more information, see the Sun manual, *Debugging a Program*.
- If you are running more than one Debugger, then ClassBrowser displays the source in all the Debuggers that are:source in all the Debuggers that are:
  - Running in the same session as ClassBrowser
  - Have the SrcDisplays Synchronization property set to Always.
- If no Debugger is running when you click on the Source button, then nothing happens.
- You can run the Debugger in any directory.
- You can query on any symbol in the Debugger window by selecting the symbol, then pressing the SourceBrowser Query button.
- When you are done displaying source in the Debugger, switch the setting in the ClassBrowser window back to Show Source in Browser. This is necessary for subsequent displays of the source of classes and functions to work correctly.

8.3 Using an External Editor with the SourceBrowser and the Debugger

The SourceBrowser and the Debugger use ToolTalk™ for communication between and among components. When the Debugger and the SourceBrowser are started with the `-editor` flag, they do not present a source pane. Instead the communication ToolTalk messages can be used by any ToolTalk-aware editor to provide an alternative displayer of source code. The ToolTalk protocol is described in the *Integrating Applications with the SPARCworks 3.0 Toolset* document and is supported by SunSoft.

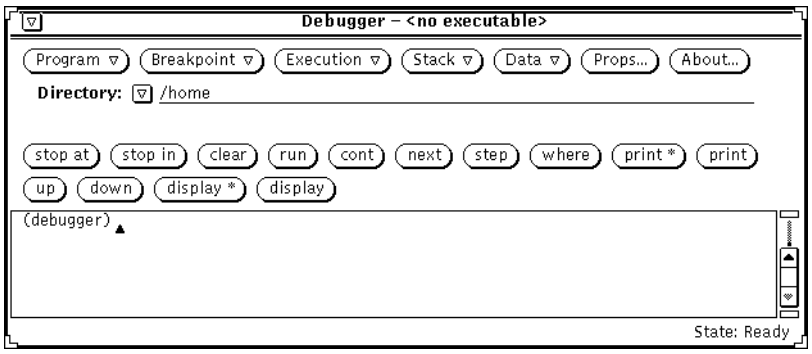
Currently SunSoft does not provide an editor that supports ToolTalk. However, externally supplied editors are available. The Xemacs™ editor and Eos™ package, available via `ftp` from the University of Illinois in `cs.uiuc.edu:/pub/era`, are used in the following descriptions.

8.3.1 Starting the Tools from the External Editor

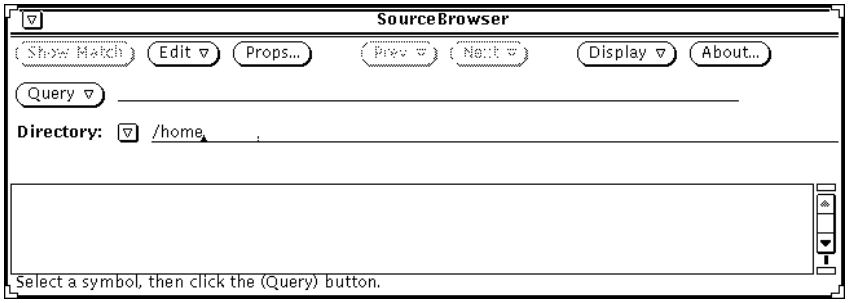
Start the Xemacs editor and instruct it to work as a SourceDisplay for the Debugger and the SourceBrowser. Read the Xemacs and Eos documentation for further details.

Start the Debugger and the SourceBrowser with the `-editor` flag. The tools with be displayed as usual except that no source pane will appear.

```
% debugger -editor &
```




```
% sbrowser -editor &
```



Other SourceBrowser components such as CallGrapher, ClassGrapher, and ClassBrowser can be invoked in the normal manner.

8.3.2 The Debugger and the External Editor

You can interact with the external editor in almost the same way as you would with the source pane in the Debugger. For example, you can apply a breakpoint in a function by selecting the name of the function and then selecting the `stop in` button. The graphics are similar in that hollow and solid arrows are also used.

The only difference between the two is in the behavior of a corner case. The Debugger determines the location for the breakpoint based on the cursor position in its source pane; the external editor uses the line of the beginning of the selection.

8.3.3 The SourceBrowser and the External Editor

You can interact with the external editor in a manner similar to how you interact with the source pane in the SourceBrowser. Source operations from the ClassGrapher, CallGrapher. or ClassBrowser behave in the same way. Queries from the main SourceBrowser window differ in that the source pane in the SourceBrowser uses glyphs to show current and non-current matches in the current and non-current queries, while the protocol used by the external editor has only enough information to present the current match of the current query. The source glyphs are identical.

Issuing More Advanced Queries 9

Chapter 3, “Issuing A Simple Query,” described how to issue a basic query for an identifier. This chapter describes how to perform more advanced queries, especially how to search for string constants.

This chapter is organized into the following sections:

Querying for String Constants	*page 153*
Using Wildcards in Queries	*page 154*
Using Strings and Wildcards Together	*page 158*
Issuing Case-Insensitive Queries	*page 159*

9.1 Querying for String Constants

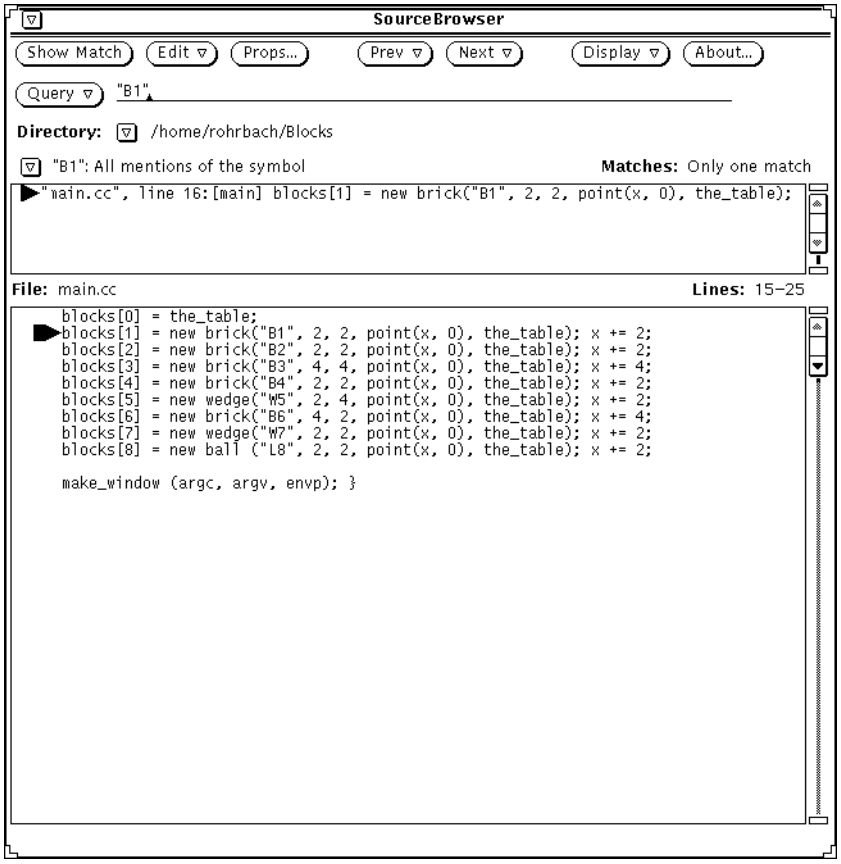
To instruct SourceBrowser to search for string constants:

1. **Select the string constant you want to search for.**
   You can type the string constant in the Text field, enclosed in quotation marks, or you can select a string constant in the source pane or in another window.

2. **Click on the Query button.**
   Or, type the string constant in the Text field and press Return.

In the following example, the user instructed SourceBrowser to find all occurrences of the block name “B1”. This block, like all other blocks in the program, is found in `main.cc`.

SourceBrowser found one occurrence of the string “B1” in main.cc



9.2 Using Wildcards in Queries

When you use wildcards to specify a search pattern, SourceBrowser gives you three options:

- Shell-style patterns
- Regular expressions
- Literal strings

To limit the number of matches you receive when including wildcards in a query, you can use the Filter command to search for items based on how they are used in a program. You can also use the Focus command to focus your search on instances of specific units of code. See Chapter 7, “Restricting the Match Set,” for details.

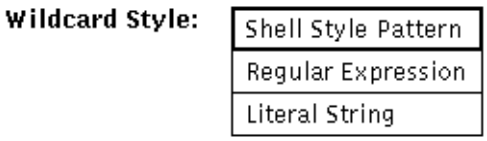
9.2.1 Shell Style Patterns

The SourceBrowser default is to search for wildcards using shell-style patterns. You may need to reset this property if you are using an already configured SourceBrowser.

See the sh(1) man page for information about shell-style pattern matching.

To issue a query using shell-style patterns:

1. **Set Wildcard Style to Shell Style Pattern in the Properties Window.**



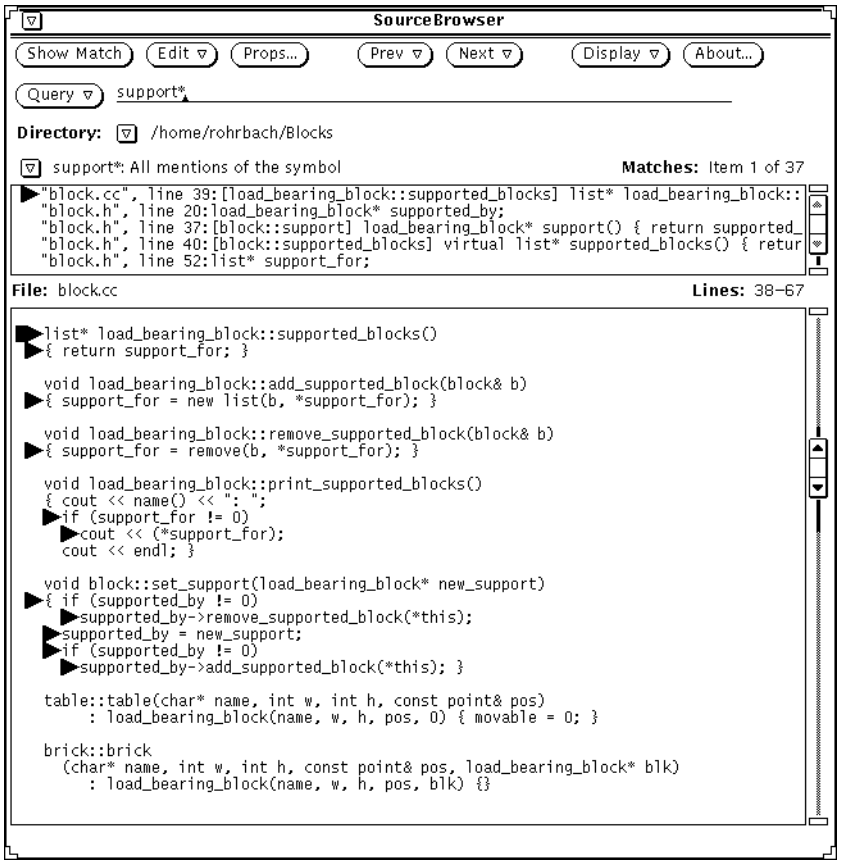
2. **Select the pattern you want to search for, then click on the Query button.**
   Or, type the pattern in the Text field and press Return.

Note – A poorly chosen pattern could return an enormous number of matches, taking a very long time. For example, “*” will match every occurance of every symbol and string in the database. In general, the earlier the “*” appears in the string, the larger the number of matches and the slower the query. Consequently, Support * is better tham S*.

In this example, the user instructed SourceBrowser to find all instances of `support*`, where "*" matches any characters. SourceBrowser displays all symbols that begin with `support`: `supported_by`, `support_for`, `supported_blocks`, and so on.

Query for support*

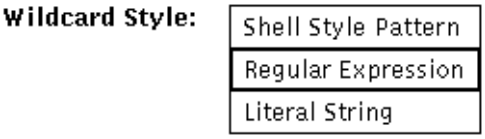
Matches all begin with the word "support".



9.2.2 Regular Expressions

To issue a query using regular expressions:

1. **Set Wildcard Style to Regular Expression in the Properties Window.**

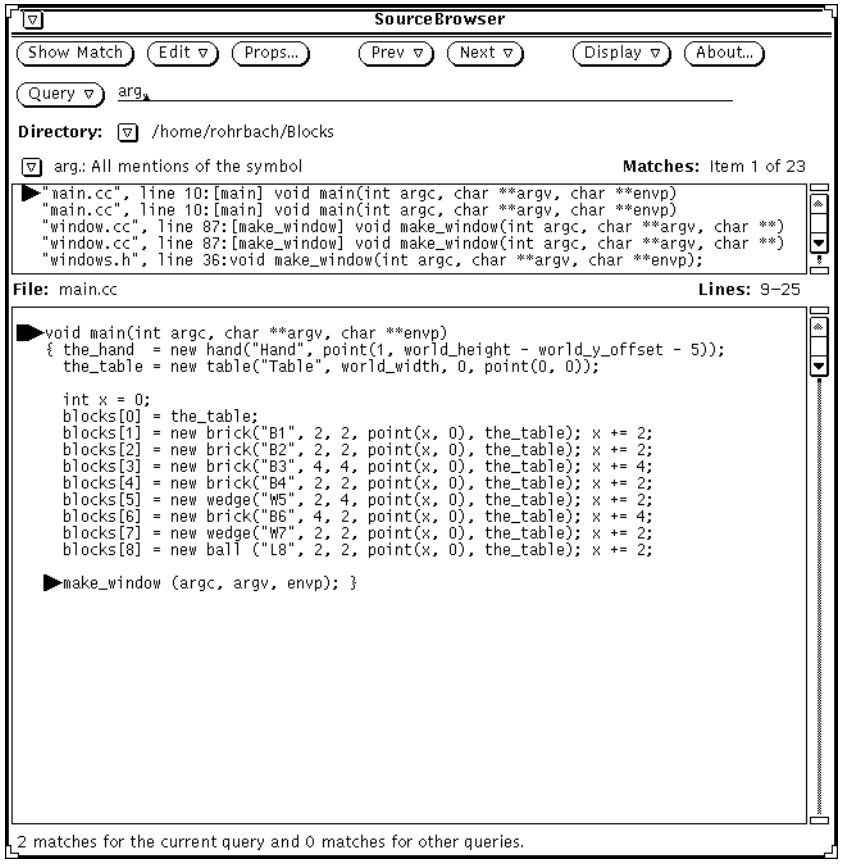


2. **Select the pattern you want to search for, then click on the Query button.**
   Or, type the pattern in the Text field and press Return.

In this example, the user instructed SourceBrowser to find all instances of `arg.` where "." matches a single character.

Query for arg.

SourceBrowser returns matches for arg1, arg2, args, argv and argc



Note that SourceBrowser automatically anchored the regular expression query. Although the user typed in `arg.`, SourceBrowser treated the query as `^arg.$`. SourceBrowser only returned matches that began with the letters "`arg`" followed by a single character. In this example, SourceBrowser returned matches for `arg1`, `arg2`, `argc`, `argv`, and `args` but would not match the argument.

If the user had explicitly anchored the regular expression using the '^' and the '$' delimiters, then SourceBrowser would have returned the same results.

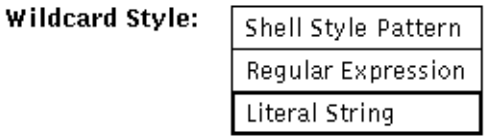
See the `regexp`(5) man page for the specification for regular expressions.

9.2.3 Literal Strings

A literal string query searches for strings that contain the special characters used in shell-style or regular expression patterns. In a literal string query, SourceBrowser ignores any special meaning such characters have.

To issue a literal string query:

1. Set Wildcard Style to Literal String in the Properties Window.



2. Select the pattern you want to search for, then click on the Query button.

For example, if you set Wildcard Style to Literal String, then instructed SourceBrowser to search for all instances of `support*` in the Blocks program, SourceBrowser would return no matches.

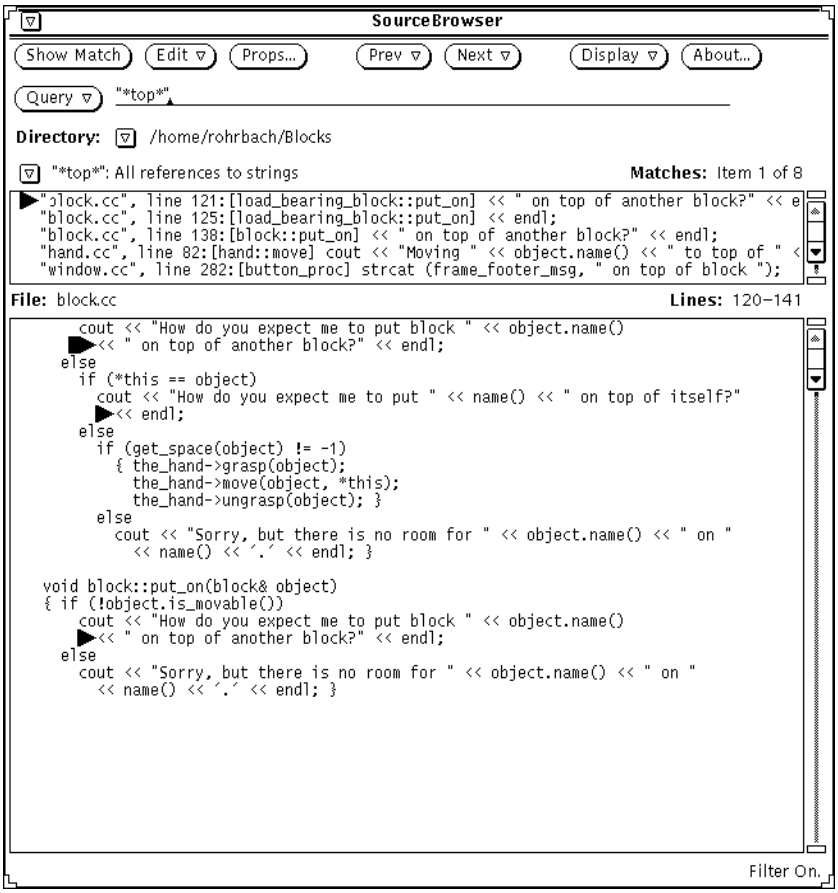
9.3 Using Strings and Wildcards Together

SourceBrowser allows you to combine strings and wildcards in your query. This feature is especially useful when searching for error messages in your program.

For example, here the user set Wildcard Style to Shell Style Pattern, then searched for every string with the word "`top`" in it. SourceBrowser returned eight matches.

Query for the string "*top*"

The string appears in the Blocks program error messages

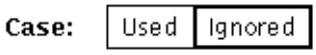


9.4 *Issuing Case-Insensitive Queries*

You can tell SourceBrowser to turn off case sensitivity when searching for symbols and strings. This is especially useful for FORTRAN programmers.

To turn off case:

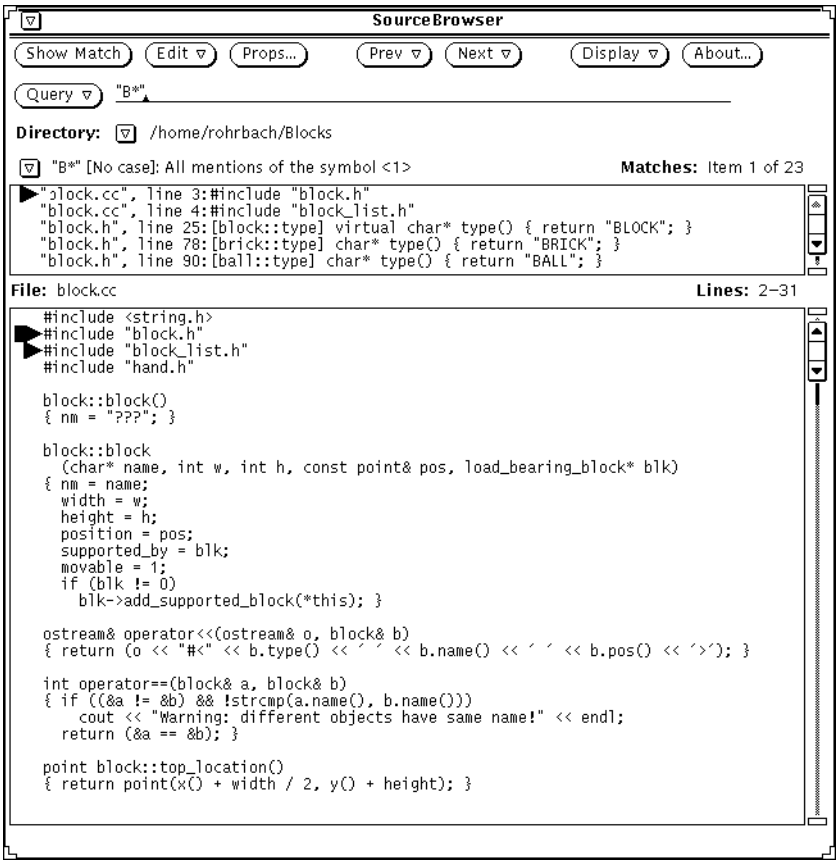
♦ **Set Case to Ignored in the Properties Window.**



Now issue your query. In this example, the user set Case to Ignored and Wildcard Style to Shell Style Pattern, then searched for all occurrences of the string “B*”. SourceBrowser returned matches for both “block.h”, “BLOCK” and so on. Note that the Query Text field includes the symbol [No case].

Text field includes [No Case] after query

SourceBrowser returns matches for both block.h and BLOCK



Customizing SourceBrowser 10

This chapter describes how to issue queries and display matches to suit your specific needs. See “Customizing the Call and Class Graphs” on page 85 for information on setting the properties in the CallGrapher and ClassGrapher. See “Customizing ClassBrowser” on page 114 to set properties for the ClassBrowser.

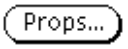
This chapter is organized into the following sections:

Setting SourceBrowser Properties	*page 162*
Understanding SourceBrowser Properties	*page 163*

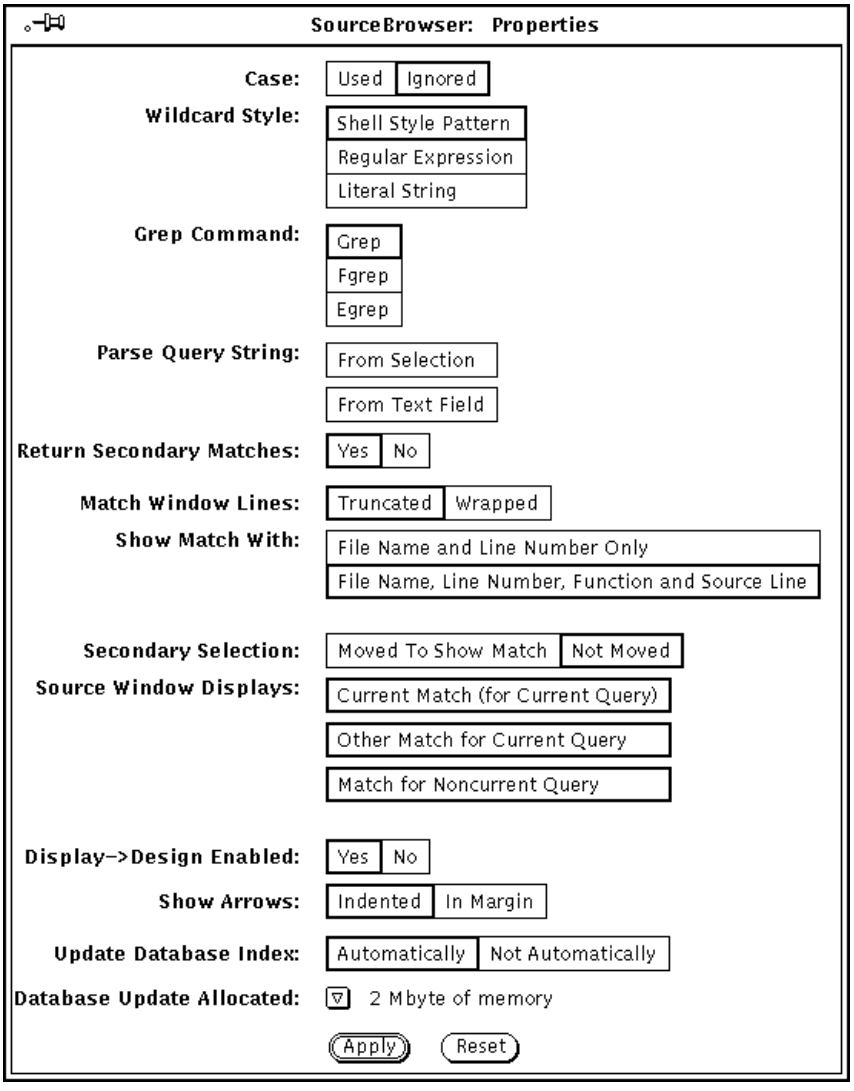
10

10.1 Setting SourceBrowser Properties

To set SourceBrowser properties:Enter the desired value or click on the desired



1. **Click on the Props button in the SourceBrowser base window.**



2. Enter the desired value or click on the desired setting, then click on the Apply button.

Any changes you make to the Properties window are retained in the `.SBdefaults` file in your home directory. All subsequent SourceBrowser sessions use the default property settings in the `.SBdefaults` file.

To discard any changes you have made in the Properties window, but have not yet applied:

- **Click on the Reset button.**
  SourceBrowser returns the properties to their last applied values. The Properties window remains open so you can make additional changes.

10.2 Understanding SourceBrowser Properties

Use the SourceBrowser Properties window to customize how you issue queries and display matches. You can also display these descriptions on-line by pointing to the property in the Properties window and pressing the Help key.

Case

The Case property tells SourceBrowser whether or not to consider the case of letters when it searches for symbols in the source code.

Used—Make a match only when the case of the symbol matches the case of the symbol in the source code. This setting is the default.

Ignored—Do not look at case when searching for symbols.

See the section “-no_case” on page 220 for information about the equivalent command-line option.

Wildcard Style

SourceBrowser provides three types of pattern matching.

Shell Style Pattern—Use shell-style patterns when issuing a query that includes wildcards. This setting is the default.

Regular Expression—Use regular expressions when issuing a query that includes wildcards.

Literal String—Ignore any meaning a special character has. This setting is useful when you want to search for a string that contains characters with special meaning in other wildcard schemes.

See the sections “-sh_pattern” on page 225, “-reg_expr” on page 225, and “-literal” on page 226 for information on the equivalent command-line options.

See “Using Wildcards in Queries” on page 154 for examples of issuing wildcard queries.

The Activate According to Pattern and Deactivate According to Pattern menu items in the Focus window also use wildcard matching. See the section “Setting the Focus” on page 130 for details.

`grep` Command

The `grep` command allows you to search your source code without having compiled or used `sbtags`. See Chapter 2 for a discussion of the `grep` command.

Parse Query String

The Parse Query String property allows you to include `sbquery` command-line options in your query.

From Selection—Interpret selected symbols with a leading dash as an `sbquery` command-line option.

From Text Field—Interpret symbols typed in the text field with a leading dash as an `sbquery` command-line option.

These settings are not mutually exclusive. Both settings are off by default.

If you turn on From Select or From Text Field properties, the Filter and Focus menu settings are ignored.

Return Secondary Matches

The Return Secondary Matches property tells SourceBrowser whether or not to display secondary matches in the match and source panes. A secondary match is an identifier inside a macro definition or reference. These matches are treated differently because SourceBrowser is unable to determine how identifiers inside macros are used.

Yes—Display secondary matches in the match and source panes. This setting is the default.

No—Do not display secondary matches. You might set this option if you are doing a lot of filtered querying and the symbols you are querying are used in a lot of macros. This action makes queries run faster and uses less memory.

See “Filtering in Macro Definitions” on page 126 for more information on secondary matches. See the section “-no_secondaries” on page 221 for information on the equivalent command-line option.

Match Window Lines

The Match Window Lines property tells SourceBrowser how to display long lines of code in the match pane.

Truncated—Truncate lines of code in the match pane. This setting is the default.

Wrapped—Wrap lines of code so that the complete line of source code is displayed.

Show Match With

The Show Match With property tells SourceBrowser the information to include in the match pane.

File Name and Line Number—Do not display the source code line and function name in the match pane. Choosing this setting causes a 50% improvement in performance when conducting queries.

File Name, Line Number, Function and Source Line—Display the source code line and function name, along with the file name and line number, in the match pane. This setting is the default.

See the section “-no_source” on page 221 for information on the equivalent command-line option.

Secondary Selection

Selection determines whether or not SourceBrowser moves the selection in the source pane when you issue a query or move between matches.

Moved to Show Match—Move the selection bar to the current match in the source pane when SourceBrowser displays a new current match. SourceBrowser highlights the current match in the source code.

Not Moved—Do not select the current match in the source pane when a new current match is displayed. SourceBrowser does not highlight the current match. This setting is the default.

Source Window Displays

The Source Window Displays property tells SourceBrowser which matches to mark with glyphs in the source pane.

Current Match (for Current Query)—Display the glyph for the current match.

Other Match for Current Query—Display the glyphs for all matches for the current query.

Match for Noncurrent Query—Display the glyphs for matches found by any query.

By default, SourceBrowser displays the glyphs for all three types of matches.

Display->Design Enabled

This property allows ToolTalk messages to be sent as a design tool. Currently Source correlation and reverse engineering are supported. See Appendix C for details on integration with design tools.

Yes - This selection causes **a Design** menu item to be added to the Display menu button. This allows you to view the design of a selected symbol in the design tool.

No - This selection disables message passing between design the SourceBrowser and the design tool. No Design menu option appears in the Display menu. Message passing is disabled by default.

Show Arrows

The Show Arrows property tells SourceBrowser where to display arrows that identify matches in the source pane.

Indented—Indent arrows so that they line up with the first character of the source line. This setting is the default.

In Margin—Place arrows in the left margin.

Update Database Index

The Update Database Index property tells SourceBrowser whether or not to automatically update the index file after you compile a program.

Automatically—Update the index file after new information has been added to the database. This setting is the default.

Not Automatically—Do not automatically update the database index. When you turn off the automatic updates, you can manually update the database index by clicking on the Update button under Edit..

See “Manually Updating the Database” on page 189 for more information.

See the section “-no_update” on page 222 for information on the equivalent command-line option.

Database Update Allocated

Press MENU on the abbreviated button to display a menu containing options for amounts of memory. Choose the amount of memory you want SourceBrowser to allocate before starting to use temporary files to build the database index. The default is 2 megabytes.

See Chapter 13, “Controlling the Database,” for more information.

See the section “-max_memory <size>” on page 220 for information on the equivalent command-line option.

Part 3—Advanced Browsing

Overview of the Database 11

This chapter provides an overview of the SourceBrowser database. See Chapter 13, “Controlling the Database”, for step-by-step instructions on how to manipulate the database.

This chapter is divided into the following sections:

Creating and Updating the Database	*page 171*
Linking Executable Files When You Build the Database	*page 177*

11.1 Creating and Updating the Database

The SourceBrowser database contains the following files:

- Browser data files created by the compiler when you compile your program with the SourceBrowser option. The browser data (`.bd`) files contain the symbols and semantic information SourceBrowser needs to browse a file. The `.bd` files are stored in the `NewRoot`, `OldRoot`, and `Refd` directory in the `.sb` (SourceBrowser) directory.
- An index file created by SourceBrowser. SourceBrowser uses the Index file to locate information in the `.bd` files.

Unless you specify otherwise, the `.sb` subdirectory is created in the current working directory from which the compiler is called. Table 11-1 describes the process.

Table 11-1 How the Database is Created

Action You Perform	What Happens in the Database
1. Compile your program with the Source-Browser option for the first time (or run the sbtags program as described in Chapter 2).	The compiler does the following: • Creates a `NewRoot` and a `Refd` directory in the `.sb` directory. • Creates a `.bd` file for each source file you compiled and stores them in the `NewRoot` directory. For a C++ program, these files have names of the form `*.cc.*.bd`. The first asterisk (*) is a placeholder for the source file name. The second asterisk is a placeholder for a string that is a hash value computed from the contents of the bd file. • Creates a `.bd` file for each header file you compiled and stores them in the `Refd` directory. These files have names of the form `*.h.*.bd`.

Table 11-1 How the Database is Created *(Continued)*

Action You Perform	What Happens in the Database
2. Issue the first query	SourceBrowser does the following: • Creates an `OldRoot` directory. • Copies the files in the `NewRoot` directory into the `OldRoot` directory. • Deletes the `NewRoot` directory. • Creates an Index file which is used to locate information in the `.bd` files. SourceBrowser creates this file while responding to your initial query.
3. Each subsequent time you compile your program with the SourceBrowser option	The compiler creates `.bd` files for files that have been modified since the last compilation. It creates a new `NewRoot` directory for the source `.bd` files. At this point, the `.sb` directory has a `NewRoot`, `Refd`, and `OldRoot` directory.
4. Issue your first query following a compilation	SourceBrowser does the following: • Copies the files in the `NewRoot` directory into the `OldRoot` directory and deletes the `NewRoot` directory. • Updates the index file before responding to your query. • Discards obsolete files.

Figure 11-1 shows how SourceBrowser databases are created in the `.sb` subdirectories in the `source1` and `source2` directories. The asterisk (*) in each `.bd` file name is a placeholder for a string that is a hash value computed

from the contents of the bd file.

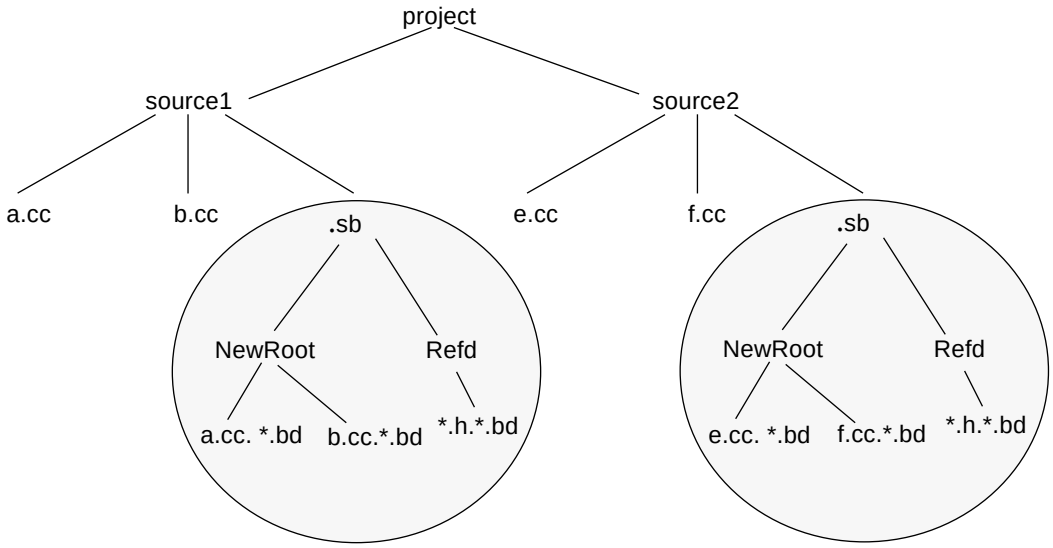


Figure 11-1 Sample SourceBrowser Database When You First Compile

Figure 11-2 shows the database after the user issued the first query.

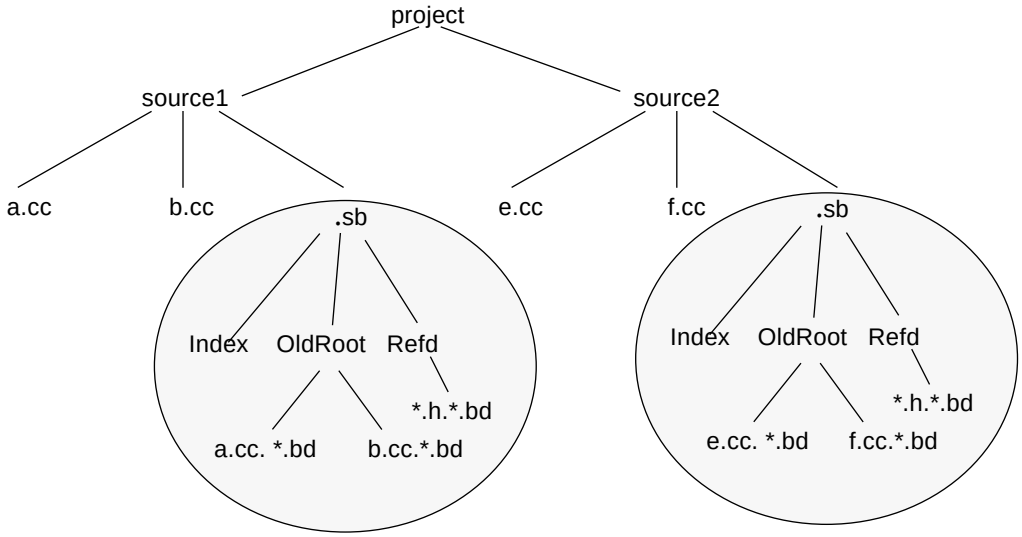


Figure 11-2 Database After First Query Issued

Now the user edited the files `a.cc` and `e.cc`. Figure 11-3 shows the results after the user recompiled the program.

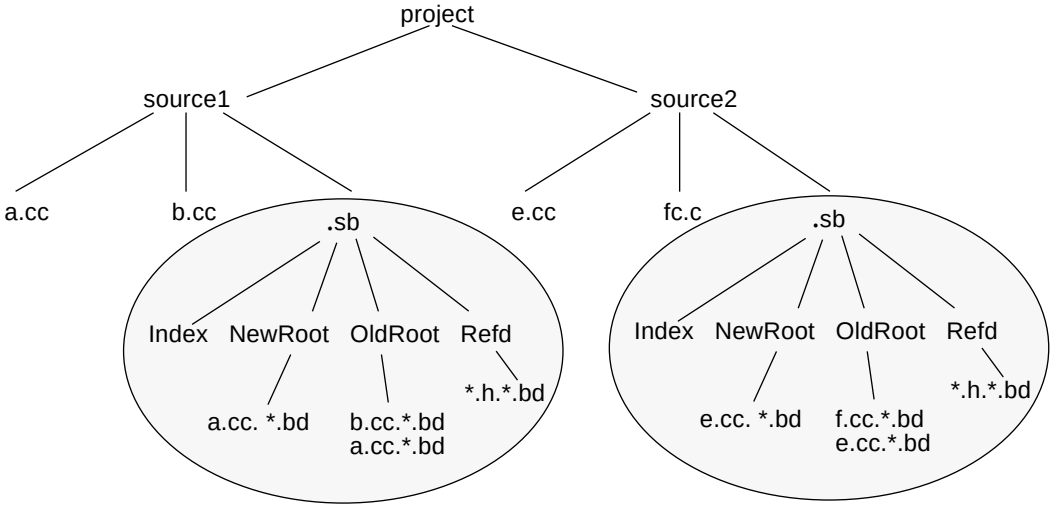


Figure 11-3 Database After Recompilation

When user issued the first query following recompilation, the database looks again as it did in Figure 11-2. The new database files a.cc.*.bd and e.cc.*.bd are now stored on the OldRoot directories and the obsolescent versions are gone.

11.2 Linking Executable Files When You Build the Database

When you link files that have been compiled with the SourceBrowser option, an additional `.bd` file is created by the linker that lists the object files used to construct the program. For example, in Figure 11-4, the linker creates an `a.out.*.bd` file.

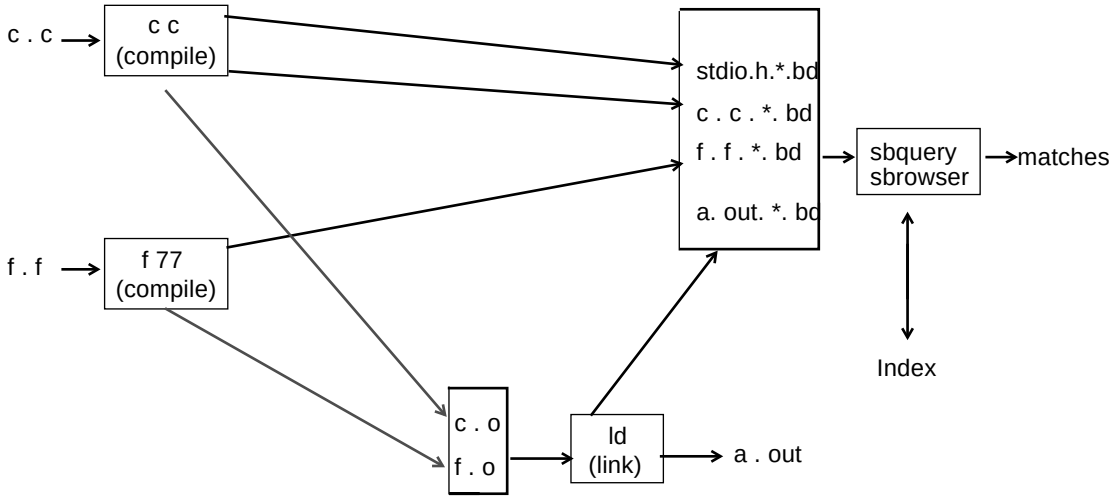


Figure 11-4 Linking and Browser Data Files

Working with Multiple Directories 12

This chapter describes how to work with source files whose database information is stored in multiple directories. By default, SourceBrowser builds the database in the current working directory. When you issue a query, SourceBrowser searches the database in the current working directory.

This chapter is organized into the following sections:

Installing a Symbolic Link between Directories	*page 179*
Consolidating Several Databases	*page 180*
Using sbinit to Browse Multiple Directories	*page 182*

12.1 Installing a Symbolic Link between Directories

To share SourceBrowser information among multiple directories, you can install a symbolic link to a common `.sb` directory from all directories in which the compiler is executed and queries issued. Using a symbolic link to create a database shared between directories is the most space-efficient sharing mechanism.

To create a symbolic link to any `.sb` subdirectory:

♦ **Execute the command**

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

Or, if you use `make`, include the following code fragment in your Makefile:

```
.INIT: .sb

.sb:

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

Note that *path* can be relative or absolute.

Figure 12-1 shows a conceptual view of installing a symbolic link to a common subdirectory. In this figure, the `.sb` directories in `source1` and `source2` are symbolically linked to the common `.sb` directory in `project`. SourceBrowser runs equally well in all three directories and returns the same result no mater which directory it uses.

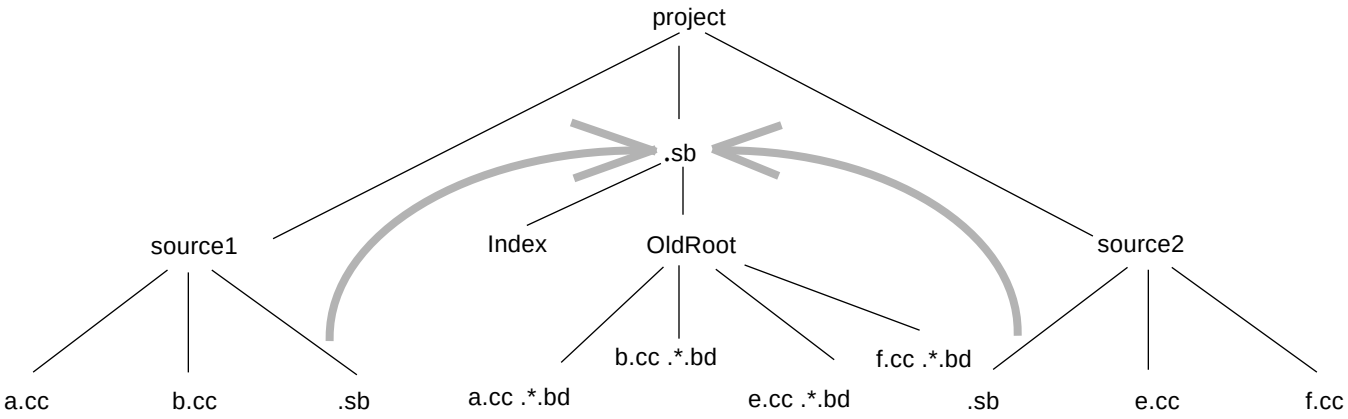


Figure 12-1 Installing a Symbolic Link

12.2 Consolidating Several Databases

You can also consolidate several SourceBrowser databases into a single database. You might do this if:

- You ran out of space on the partition where you keep your database, and you want to move it to another partition.

To consolidate several databases into a single database:

1. **Change directory to your project directory.**
2. **Use the `tar` command to move the `.sb` subdirectories in your project directory up one level.**

3. **Remove the old** `.sb` **subdirectories.**

4. **Create symbolic links to the** `.sb` **directory in your project directory.**

5. **Remove the Index file in the destination** `.sb` **directory.**

For example, in Figure 12-2, SourceBrowser databases are created in the .sb subdirectories in the `source1` and `source2` directories.

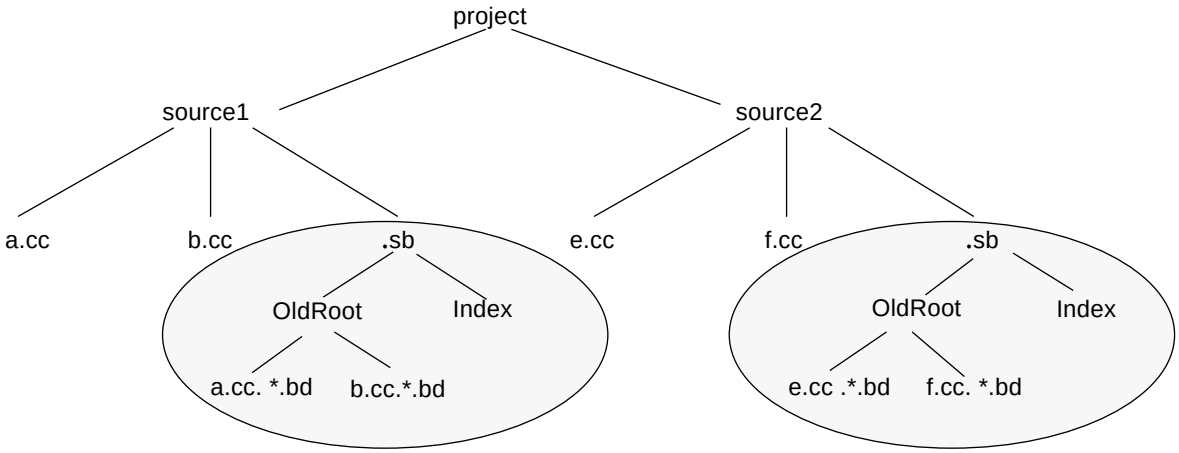


Figure 12-2 Database before Consolidation

In this example, the user moved the database in the `.sb` directories in `source1` and `source2` to the `project` directory.

```
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
```

Figure 12-3 shows the results.

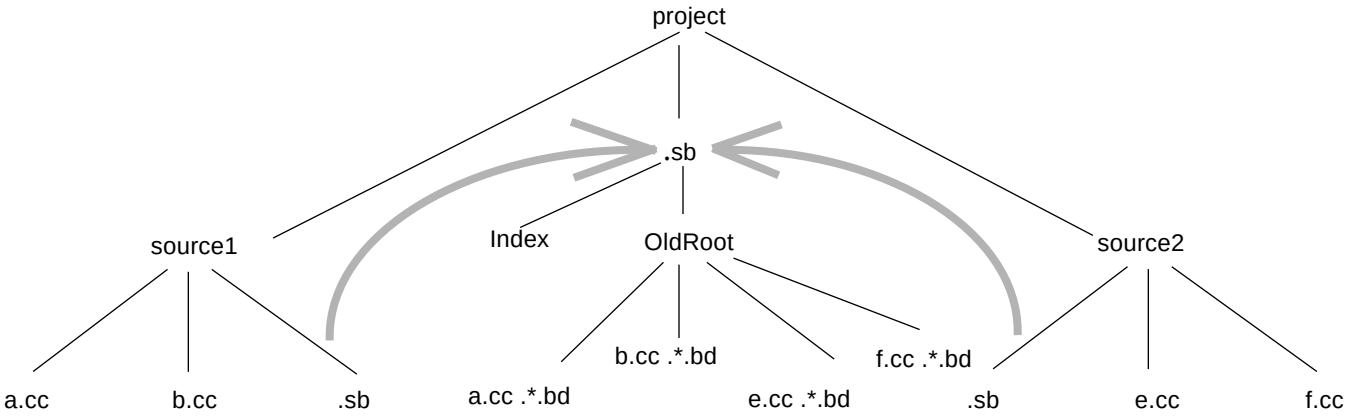


Figure 12-3 Database After Consolidation

12.3 Using sbinit to Browse Multiple Directories

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

- `import`—Reads databases in directories other than the current working directory.
- `export`—Writes database component files associated with specified source files to directories other than the current working directory of the compiler.
- `automount-prefix`—Creates `.bd` files for source files you access with the automounter. You can access these files after the automounter has unmounted the sources.
- `replacepath`—Specifies how to modify path names in the SBrowser database.

The SourceBrowser and the compiler default is to look in the current working directory for the `.sbinit` file.

12.3.1 Reading Databases in Other Directories

To read a database in a directory other than the current working directory:

- **Add** `import` *path* **to your** `.sbinit` **file.**

  path is the path to the `.sb` subdirectory that contains the database you want to import.

In contrast to installing a symbolic link, which creates a common database directory, the `import` command enables you to retain separate databases for separate directories.

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 each need to maintain their own SourceBrowser database, both of which can then be read but not written by programmers working on the other project.

In Figure 12-4, the current working directory is `/project/source1`. The user instructed SourceBrowser to read the SourceBrowser database in source2 by including either of these commands in the source1 `.sbinit` file:

```
import /project/source2
```

or

```
import ../source2
```

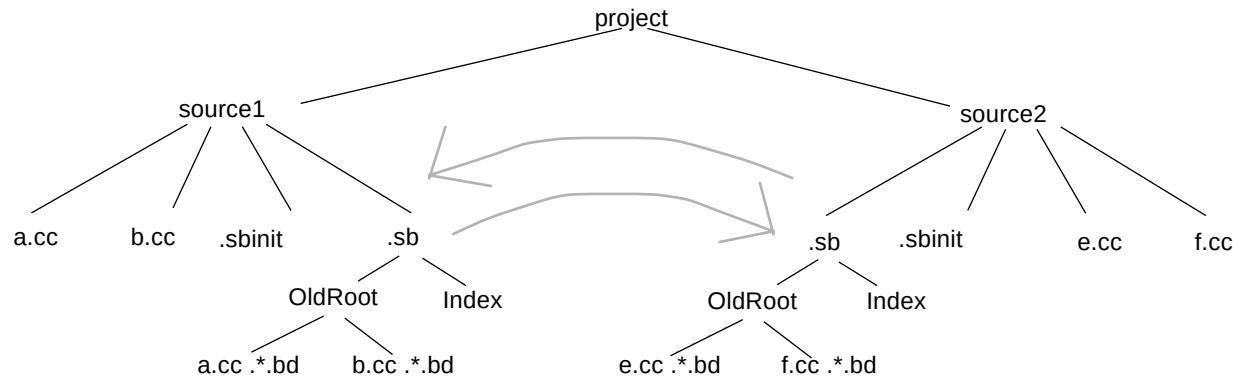


Figure 12-4 Importing Databases in Other Directories

12.3.2 Writing to Databases in Other Directories

To write to a database in a directory other than the current working directory:

- **Add** `export` ***prefix*** `into` ***path*** **to your** `.sbinit` **file.**

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

The `export` command 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 your `.sbinit` files include multiple `export` commands, then you must arrange them from the most specific to the least specific. The compiler scans `export` commands in the same order that it encounters them in the `.sbinit` file.

In Figure 12-5, to place the `.bd` file and index file created for files from `/usr/include` in a `.sb` subdirectory in the `sys` subdirectory, the user included this `export` command in the `.sbinit` file for `source1`:

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

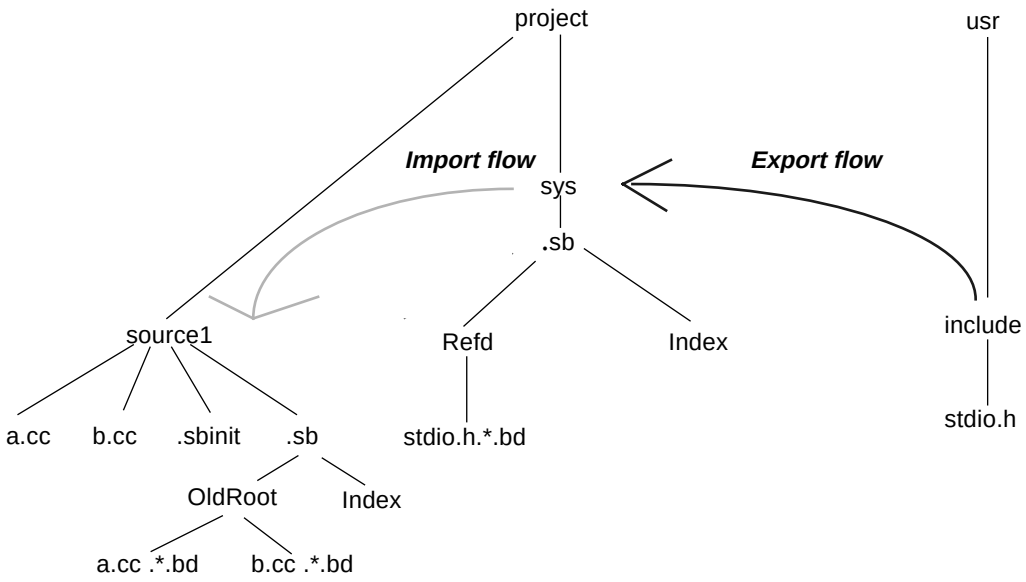


Figure 12-5 Exporting Shared System Files

If the configuration had included a `source2` directory with a `.sbinit` file containing the same `export` command shown in Figure 12-5, then 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 `.bd` files created for source files not explicitly mentioned by an `export` command in the current working directory. In Figure 12-5, 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, an implied `import` command causes SourceBrowser to read each database that you have instructed the compiler to create. Based on the configuration in Figure 12-5, SourceBrowser searches in the database in the `sys` subdirectory, as well as in the database in the `source1` directory, each time you issue a query.

As another example, in Figure 12-6, to place the `.bd` file and index file created for files from `/project/include` in the `.sb` subdirectory in the `/project/include` subdirectory, the user included this export command in the `.sbinit` file for source1:

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

An implied `import` command causes SourceBrowser to read the database in `/project/include`.

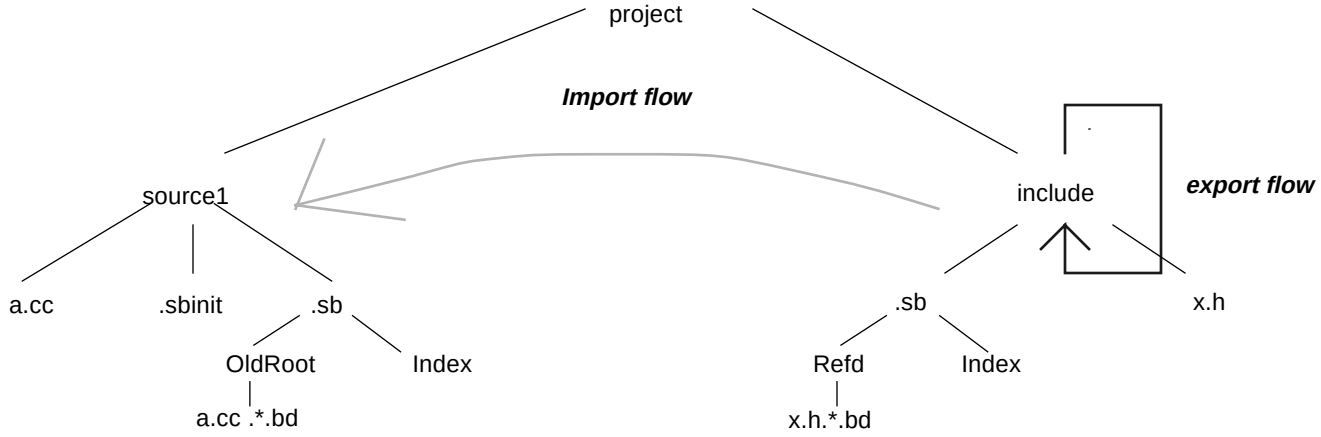


Figure 12-6 Exporting Shared Include Files

In Figure 12-6, SourceBrowser stores the `.bd` files in a common subdirectory even though it located the `include` file in a nonstandard location. The `export` command saves disk space if the project includes multiple references from many different directories to the same `include` file.

Because the `export` command includes an implied `import` command, SourceBrowser reads the database in the `include` subdirectory, as well as the database in the `source1` subdirectory, each time the user issues a query.

12.3.3 Browsing on Multiple Machines

The `automount-prefix` command enables you to run SourceBrowser on a machine other than the one on which you compiled your program. For example, suppose the machine `bird` is a fast server. You may want to compile the programs on `bird` and then browse the files on your own system.

At first glance, this may not seem possible; the browser database that is created when you run the compiler contains the absolute path for each source file. If the absolute path is not uniform across machines, then SourceBrowser will not be able to display the source files when it responds to a query.

To get around this problem, you can do either of the following:

- Ensure that all source files are mounted at the same mount point on all machines.
- Compile your programs in an automounted path. A reference to such a path causes the *automounter* to automatically mount a file system from another machine.

To run SourceBrowser in an automounted path:

♦ **Add `automount-prefix` *mount_point trigger_point* to your `.sbinit` file.**

The `automount-prefix` command defines which automounter prefixes to remove from the source names stored in the database. The automounter mounts the file systems under the directory *mount_point*. The default is `/tmp_mnt`. The prefix you supply to the pathname to access the exported file system is *trigger_point*. The default is `/net`.

For more information on using the automounter, see the `automount`(1M) man page.

12.3.4 Using `replacepath`

The replacepath command has the form:

> **`replacepath`** *from-prefix to_prefix*
>
> where *from-prefix* is the prefix path that will be replaced by *to-prefix* whenever it is found.

The following default **`replacepath`** command is used to strip away automounter artifacts:

> **`replacepath`** */tmp_mnt /net*
>
> When used for this purpose, the command should be given as the first argument and the *mount_point* and *trigger_point* given as the second argument.

12.3.5 If You Cannot Find the .sbinit File

The SourceBrowser default is to look in the current working directory for the `.sbinit` file.

To instruct SourceBrowser and the compiler to search for the `sbinit` file in another directory:

- **Set the environment variable** *SUNPRO_SB_INIT_FILE_NAME* **to** ***/absolute/pathname/*`.sbinit`**.**

In this example, the user set *SUNPRO_SB_INIT_FILE_NAME* to `/net/dynamo/export/home/dynamo1/browser/examples/.sbinit`.

```
venus% setenv SUNPRO_SB_INIT_FILE_NAME
/net/dynamo/export/home/dynamo1/browser/examples/.sbinit
```

Controlling the Database 13

This chapter describes advanced techniques for building the database. This chapter is organized into the following sections:

Manually Updating the Database	*page 189*
Accessing a Locked Database	*page 190*

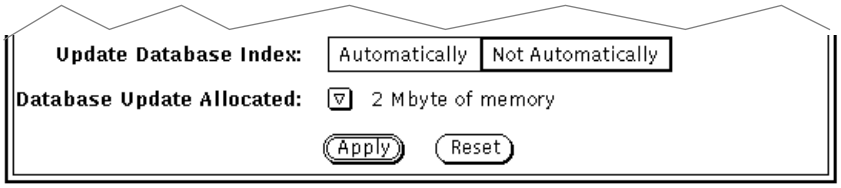
See Chapter 11, “Overview of the Database,“ for an introduction of the SourceBrowser database.

13.1 Manually Updating the Database

By default, SourceBrowser automatically updates the database before the initial query. However, if you have a large database and you compile your program often, then the update can be very slow. In this case, you may want to turn off the automatic updates, then manually update the database when you have more time.

To turn off the automatic updates:

♦ **Set Update Database Index to Not Automatically in the SourceBrowser Properties window, then click on the Apply button.**



To manually update the database:

♦ **Choose Update Database from the Edit menu in the SourceBrowser main window.**

13.2 Accessing a Locked Database

If the update of the index file is aborted for some reason (for example, a power failure causes your machine to crash), then the next time you issue a query, SourceBrowser may display a message telling you that the database is locked. By default, SourceBrowser attempts to access the locked database 100 times before giving up.

To instruct SourceBrowser to attempt to unlock the database more than or fewer than 100 times:

♦ **Set the environment variable** *SUNPRO_SB_ATTEMPTS_MAX* **to the desired integer value.**

In this example, the user set *SUNPRO_SB_ATTEMPTS_MAX* to `10`.

```
venus% setenv SUNPRO_SB_ATTEMPTS_MAX 10
```

SourceBrowser locks the database when it updates the index file to prevent a second SourceBrowser from updating the file at the same time. During normal operation, the second SourceBrowser waits for the first SourceBrowser to remove the lock before proceeding. If the first SourceBrowser fails during the update, then the database remains permanently locked.

Part 4—Appendixes

Troubleshooting A

This appendix describes how to resolve problems with SourceBrowser. It is organized into the following sections:

Problem Checklist	*page 195*
Property Default Values	*page 198*
Reporting Problems	*page 200*
SourceBrowser Messages	*page 201*

A.1 Problem Checklist

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

- `sbrowser` is installed in the standard location. Check with your system administrator.
- `sbrowser` can be found in your **PATH**. Use the `which`(1) command to check if `sbrowser` is in your path.
- The *SUNPRO_SB_EX_FILE_NAME* environment variable is *not* set. SourceBrowser relies on finding the file `sun_source_browser.ex` in the `lib` directory next to the directory that contains `sbrowser`. If you install in a nonstandard way so that the `sun_source_browser.ex` file is not found where expected, the system will return an error message when you try to run SourceBrowser.

To instruct SourceBrowser to search for the `sun_source_browser.ex` file another directory:

Set the environment variable *SUNPRO_SB_FILE_NAME* to */absolute/pathname/*`sun_source_browser.ex`

In the following example, *SUNPRO_SB_FILE_NAME* is set to `/net/dynamo/export/home/browser/sun_source_browser.ex`

```
venus% setenv  SUNPRO_SB_FILE_NAME
/net/dynamo/export/home/browser/sun_source_browser.ex
```

- The *SUNPRO_SB_INIT_FILE_NAME* environment variable is *not* set. SourceBrowser relies on finding the `.sbinit` file in the current working directory. You may run into problems if SUNPRO_SB_INIT_FILE_NAME points to the wrong file. See “Using sbinit to Browse Multiple Directories” on page 182 for details.
- You are using the correct environment variables. All environment variables for the 3.0 release begin with the prefix SUNPRO_. Environment variables from previous releases are not compatible with this release.
- The SourceBrowser database is corrupt. To determine if the database is corrupt, run `sbquery`. If you get an error message, then quit SourceBrowser and remove the database. Recompile your program with the SourceBrowser option and restart the tool.
- The SourceBrowser database is locked. Look at the `db_` messages to determine if the SourceBrowser database is locked. If it is, then you can either issue the sbquery command with the `-break_lock` option or remove the `.sb` directory and recompile your program. See Chapter B, “ for a description of the `-break_lock` option.

  The compilers you are using are from the same release level. If you get messages about version mismatches, then remove and rebuild the database. If the problem persists, then you need to have the compilers upgraded to the same level. Contact your system administrator.

  Make sure that the compilers and the SourceBrowser you are using have the same release level. SourceBrowser Version 3.0, and later, must be used with SPARCcompilers 3.0, or later. Some incidental incompatibility exists. Releases 3.0 and 3.0.1 are compatible.

- Make sure that you are not intermixing SPARC and x86 databases. The SPARC and x86 versions of the SourceBrowser are incompatible. A SourceBrowser database generated on x86 cannot be used by a SPARC SourceBrowser, and vice versa. The incomaptiblity between the two may cause a Segmentation violation which, in releases prior to 3.0.1, will not be detected.
- Your `.sbinit` file contains the proper commands.

  Check whether you have an overly complicated `.sbinit` file. This overload may happen if, for each subdirectory of the project, you have a `.sbinit` file that imports databases from the other directories. For example, if you have 20 subdirectories, then the `.sbinit` file for each subdirectory has 19 `import` commands. This overload increases the time it takes SourceBrowser to perform a query. To improve performance, create a symbolic link to a common `.sb` directory. See "Installing a Symbolic Link between Directories" on page 179 for details.

  On the other hand, you may have too many symbolic links in a project that spans directories. Replace the symbolic links with a single `.sbinit` file. See "Using sbinit to Browse Multiple Directories" on page 182 for details.
- Your window system does not have enough resources. If SourceBrowser cannot activate one of its popup windows, then the window system may be running out of resources. Contact your system administrator for help.
- You have chosen the desired settings in the Properties window. In particular, check the Case setting. The SourceBrowser default is to make a match *only* when the case of the symbol matches the case of the symbol in the source code. Also, if you set Show Match With in the Properties window to File Name, Line Number, Function and Source Line, the speed of the query is 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 File Name and Line Number Only. See Chapter 10, "Customizing SourceBrowser," for details. The default property settings are listed the following section.
- If the settings in your Properties window seem unusual, go to your home directory and remove the `.SBdefaults` file, then exit all SourceBrowsers you are running. The `.SBdefaults` file retains all changes you make to the Properties window and passes them on from session to session. When you exit the window system, SourceBrowser returns to the factory-set default property settings.

- If the Source button in either the CallGrapher, ClassGrapher, or ClassBrowser appears broken, then check the following:
  - The Show Source In property in the CallGrapher, ClassGrapher, or ClassBrowser is set to Debugger.
  - The SrcDisplay Synchronization property in the Debugger Window Configurations Properties window is set to Always.

  If both properties are properly set, but no Debugger is running in the current session, then nothing happens when you click on the Source button. See for details on using the Debugger with the Graphers. See "Displaying Source in the Debugger" in Chapter 8, for information on using the Debugger with ClassBrowser.
- You properly set the Filter or Focus. Check the SourceBrowser right footer to determine whether the Filter or Focus are set. You may have issued a query outside the range of the Filter or Focus. See Chapter 7, "Restricting the Match Set," for details.
- If SourceBrowser does not accept the Filter or Focus you set, then check the Parse Query String setting in your Properties window. If either From Selection or From Text Field is turned on, then SourceBrowser ignores the Filter and Focus. Instead, SourceBrowser expects you to do your filtering and focusing using command-line options in the Text field.

A.2 Property Default Values

This section lists the factory-set property values for the SourceBrowser base window, CallGrapher, ClassGrapher, and ClassBrowser.

Table 1: SourceBrowser Base Window Default Properties

Property	Default
Case	Used
Wildcard Style	Shell Style Pattern
Grep Command	grep
Parse Query String	No options are selected
Return Secondary Matches	Yes
Match Window Lines	Truncated

Table 1: SourceBrowser Base Window Default Properties

Property	Default
Show Match With	File Name, Line Number, Function, and Source Line
Secondary Selection	Not Moved
Source Window Displays	All three options are selected
Display->Design Enabled	No
Show Arrows	Indented
Update Database Index	Automatically
Database Update Allocated	2 Mbytes of memory

Table 2: CallGrapher Default Properties

Property	Default
Expansion Depth	1
Double-click	Source, Calls, and Called By
Graph Orientation	Horizontal
Function Names	Short
Selection	Not Moved
Show Source In	Debugger
Arrowheads	On
Contraction Radius	1

Table 3: ClassGrapher Default Properties

Property	Default
Expansion Depth	1
Double-click	Source, Derived Classes, and Base Classes
Graph Orientation	Horizontal
Selection	Not Moved

Table 3: ClassGrapher Default Properties

Property	Default
Show Source In	Debugger
Arrowheads	On
Contraction Radius	1

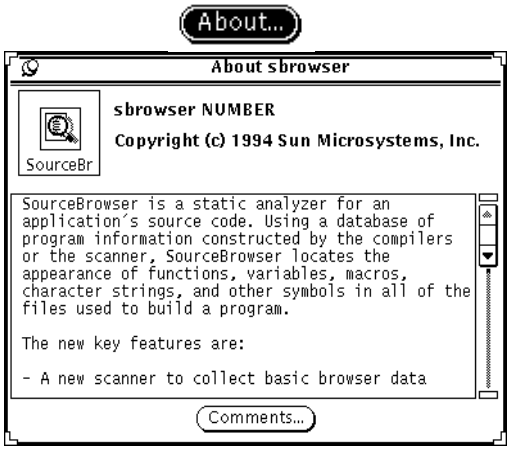
Table 4: ClassBrowser Default Properties

Property	Default
Display Options	Private, Protected, and Public Members
Show Glyphs	Virtual
Graph Mode	Manual
Show Source In	Debugger
Maximum History	10

A.3 Reporting Problems

If you have gone through the checklist and still have problems using SourceBrowser, then call SunSoft at 1-800-USA-4SUN or call your local service office. Have the SourceBrowser version number ready to give to the dispatcher.

To obtain the SourceBrowser version, select the About button from the SourceBrowser. Version information is displayed.



A.4 SourceBrowser Messages

SourceBrowser displays messages to provide you with information or tell you about an error. This section lists the SourceBrowser messages and offers instructions about what to do to correct the problem.

You can also display these help messages on-line by pointing to the error message and pressing the Help key.

If SourceBrowser has an internal error, then ask your system administrator for help.

`All mentions of the symbol.`

SourceBrowser has found all occurrences of the symbol in the source code. To limit the number of matches you receive, you can use the Filter command to search for items based on how they are used in a program. You can also use the Focus command to focus your search on instances of specific units of code.

n matches for this source line.

n symbols matched the query on this line in the source code.

SourceBrowser can only display one glyph per line with match(es).

Attempt to query for the empty string.

SourceBrowser cannot conduct a query because you have not specified a symbol to search for.

You can type the symbol in the Text field, or you can select a symbol in a SourceBrowser window or in another window.

Cannot store resource database. Failed to create default file.

SourceBrowser cannot store the changes you made in the Properties window in your .SBdefaults file.

Cannot store resource database. No write access.

SourceBrowser cannot store the changes you made in the Properties window in your .SBdefaults file. You do not have permission to write to the file.

Change of working directory failed.

The path name you typed in the notification window does not exist.

Type the correct path name in the text field. You can enter either a relative or absolute path name.

If you are returning to a previous directory, then you can simply choose the directory name from the directory history menu.

Could not create Filter window.

SourceBrowser cannot create additional windows because your window system has run out of resources. Ask your system administrator for help.

Could not create Focus window.

SourceBrowser cannot create the Focus window because your window system has run out of resources. Ask your system administrator for help.

`Could not create Property window.`

SourceBrowser cannot create the Properties window because your window system has run out of resources. Ask your system administrator for help.

`Could not load file` *`filename.`*

SourceBrowser could not find the source file that contains the match.

Check if the source files have been moved to a different directory. Or, if you are running SourceBrowser on a machine other than the one on which you compiled your program, then check if the source files are mounted on the correct path.

If the source files are in the directory in which they were compiled, and SourceBrowser still cannot load the file, then your window system may be running out of resources. Ask your system administrator for help.

`Filter On.`

SourceBrowser applied the current filter to your query.

To determine if your query will be filtered, check the SourceBrowser right footer. The footer displays a message when filtering is turned on.

To see the current filter setting, activate the Filter window.

`Focus On.`

SourceBrowser applied the current focus to your query.

To determine if your query will be focused, check the SourceBrowser right footer. The footer displays a message when focusing is turned on.

To see the current focus setting, activate the Focus window.

`Filter and Focus On.`

SourceBrowser applied the current focus and filter to your query.

To determine if your query will be focused or filtered, check the SourceBrowser right footer. The footer displays a message when filtering or focusing is turned on.

To see the current filter setting, activate the Filter window. To see the current focus setting, activate the Focus window.

`Loading the focus panel, be patient...`

SourceBrowser is currently loading the Focus panel with all available items for the unit of code you selected. This unit contains many items. You cannot perform any other browser actions while SourceBrowser loads the Focus panel.

`No database in this directory.`

SourceBrowser cannot find the `.bd` and index files in this directory.

Change to the directory in which you compiled your files. First, click on the current directory name; then, enter the new path name in the Directory text field and press Return.

You may get this error message if you started SourceBrowser in your home directory and forgot to move to the directory in which you compiled your program.

`No filter available.`

SourceBrowser cannot display the Filter window because either the database or the `sun_source_browser.ex` file is inconsistent.

First try quitting SourceBrowser, rebuilding the database, and then running SourceBrowser again.

If that does not work, then ask your system administrator to check the installation of SourceBrowser to make sure you have the proper `sun_source_browser.ex` file.

`No filter defined in this environment.`

SourceBrowser cannot display the Filter window because either the database or the `sun_source_browser.ex` file is inconsistent.

First try quitting SourceBrowser, rebuilding the database, and then running SourceBrowser again.

If that does not work, then ask your system administrator to check the installation of SourceBrowser to make sure you have the proper `sun_source_browser.ex` file.

`No focus units of that type are available.`

No focus units are available from the type of code you specified. The Focus window is empty.

Specify a different type of code on which to focus.

`No lines to activate`

SourceBrowser cannot activate items in the Focus window because you did not select any items or you selected an item outside the Focus window.

Check that you made a selection or that your selection is correct before you activate items using the Activate menu.

`No lines to deactivate.`

SourceBrowser cannot deactivate items in the Focus window because you did not select any items or you selected an item outside the Focus window.

Check that you made a selection or that your selection is correct before you deactivate items using the Deactivate menu.

`No matches.`

SourceBrowser cannot find any occurrences of the symbol, string constant, or search pattern you specified.

Either no such symbol exists in the source code, or you specified a filter or a focus that excluded all occurrences of the symbol.

Change either the symbol, filter, or focus. Try including wildcards in your search pattern if you are not sure how to spell the symbol.

`No match is available.`

The current query returned no matches. Choosing any item from either the Next or Prev menus does not apply for this match.

Make sure you typed in the symbol correctly. Also make sure you specified a filter or focus that includes the symbol.

`No other directory.`

All source files that contain matches for this query are in the current working directory. Choosing Directory from the Next or Prev menu does not apply for this match.

`No such match.`

SourceBrowser cannot find any occurrences of the symbol, string constant, or search pattern you specified.

Either no such symbol exists in the source code, or you specified a filter or a focus that excluded all occurrences of the symbol.

Change either the symbol, filter, or focus. Try including wildcards in your search pattern if you are unsure how to spell the symbol.

`No symbol provided.`

SourceBrowser cannot conduct a query because you did not provide a symbol to search for in the Text field.

If Parse Query String is set to From Text Field, then make sure that you include a symbol after the command-line arguments.

`Nothing matched pattern.`

SourceBrowser did not activate or deactivate any items in the Focus window because no items matched the search pattern you specified.

First check that the Wildcard setting in the Properties window is correct. If it is correct, change your wildcard pattern, then reissue the Activate/Deactivate According to Pattern command.

`Rearranging the database, please be patient...`

SourceBrowser is currently updating the `.bd` and index files in the database.

If you have a large database and you compile your program often, then the update can be very slow. You cannot perform any other browser actions during the update. In the future, you may want to turn off the automatic updates, then manually update the database when you have more time.

To turn off the automatic updates, set Database Update Allocated to Not Automatically in the Properties window.

To manually update the database, choose Update Database from the Edit menu.

`Select a symbol, then click the (Query) button.`

To issue the first query of a session, first select the identifier you want to search for. The identifier can be an identifier you have typed in the Text field, or an identifier in another window. Then, click on the Query button.

If you are unsure of what to query for, then begin by selecting a symbol that is central to your program. For example, if your program is written in ANSI C or C++, then start by querying for the symbol `main`. For other languages, choose a symbol used early in the program.

`The database in %s is locked, retrying every 10 seconds.`

The SourceBrowser database in pathname <%s> is locked. You cannot update the index file.

Another user may be updating the database. The database remains locked until the other user finishes the update.

If the database is locked for some other reason (for example, a power failure aborted an update), then you can either issue the `sbquery` command with the `-break_lock` option or remove the `.sb` directory and recompile.

`The selection must be in the match or source window.`

Before you choose Selected Match from the View menu, you must select a match in either the match or source pane. Making a selection in any other window will not work.

To select a match, click on any line in the match pane or a line with an arrow in the source pane. Use the scroll bars on the side of the window to scroll through the pane if necessary.

A.4.1 CallGrapher and ClassGrapher

`Graph: `*`n`*` nodes, `*`m`*` edges`

The current graph has *n* nodes and *m* edges.

In the CallGrapher, a node is a name that represents a function. An edge is a line that represents a function call.

In the ClassGrapher, a node represents a class. An edge represents class inheritance.

`Node already in graph`

The node you tried to add is already in the graph.

If the node is not in the current display, then the Grapher scrolls the graph so that the node is in the center of the display.

`Node not found`

The Grapher could not find the node you selected.

If you typed the node name in the Grapher Text field, then check that you spelled the name correctly.

You can add the node to the graph by selecting the node name, then clicking on the Add Node button.

`No graph additions`

The Grapher made no changes to the call or class graph.

Either you called up and empty graph, attempted to expand a node that has no additional expansion levels, or tried to add a node that does not exist.

If you are adding a node, then check that the name you typed in the Text field is a valid function name.

`No graph node selected`

You clicked on the Source or Browse button without selecting a node on which to operate.

To select a node in the graph, click anywhere on the node name. If you select more than one node, then the Grapher operates on the last node you selected.

`Nothing selected`

You clicked on the Query button without selecting a node to query on.

To select a node in the graph, click anywhere on the node name. Or, you can type and select the node name in the Grapher Text field. If you select more than one node, then the Grapher performs successive queries.

`No text selected`

The Grapher could not perform the add or find node operation because you did not specify a node name.

Type and select the node name in the Grapher Text field, then repeat the add or find node operation.

`Showing source in Debugger...`

Show Source In is set to Debugger in the Grapher or ClassBrowser Properties window. When you click on the Source button, SourceBrowser attempts to show the source of the selected node or class in the Debugger. If nothing happens:

- Ensure that you started both the Debugger and SourceBrowser from the same Manager.
- Ensure that the program you are browsing is properly loaded into the Debugger.
- Ensure that SrcDisplays Synchronization is set to Always in the Debugger Window Configurations Properties window.

To display the source in SourceBrowser, set Show Source In to Browser in the Grapher or ClassBrowser Properties window.

A.4.2 ClassBrowser

`Automatically bring up ClassGrapher...`

ClassGrapher automatically graphs a class in the ClassGrapher window when it becomes the currently browsed class. The ClassBrowser Graph Mode property is set to Auto.

If you do not want to automatically graph the current class, then open the ClassBrowser Properties window and set Graph Mode to manual. You must then use the Graph menu in the ClassBrowser base window to graph a class.

`Invalid class name: name`

ClassBrowser cannot find the class *name*.

Either no such class exists in the source code, or you specified a filter or a focus that excluded all occurrences of the class.

Choose Class List from the Browse menu to find out all defined classes. Change either the class name, filter, or focus. Try including wildcards in your search pattern if you are unsure how to spell the class name.

`Invalid selection`

The symbol you selected to display the source of is invalid. The valid symbols for showing source are: class name, member functions, and friend functions. If you select a symbol (e.g. variable) from Data Members, you may receive an error message. Correct your selection, then re-click on the Source button.

`Please select/specify a class name to browse`

You clicked on the Browse button without specifying a class to browse.

You can type the class name in the Text field and press Return. Or, you can select a class anywhere in the ClassBrowser or another window and click on the Browse button.

If you are unsure of the classes in your program, then choose Class List from the Browse menu. ClassBrowser displays a window with an alphabetical list of all classes in your program. Select a class name in the List window, then click on the Browse button.

`Query the current class...`

You clicked on the Query button without selecting a symbol to query on. By default, ClassBrowser issued a query on the current class.

`Showing source in Debugger...`

Show Source In is set to Debugger in the Grapher or ClassBrowser properties window. When you click on the Source button, SourceBrowser attempts to show the source of the selected node or class in the Debugger. If nothing happens:

- Ensure that you started both the Debugger and SourceBrowser from the same Manager.

- Ensure that the program you are browsing is properly loaded into the Debugger.
- Ensure that SrcDisplays Synchronization is set to Always in the Debugger Window Configurations Properties window.

To display the source in SourceBrowser, set Show Source In to Browser in the Grapher or ClassBrowser Properties window.

`Source not available.`

ClassBrowser cannot find the source code of the class, member function, or friend function you selected.

Check that the name you selected is a valid symbol.

If you selected a valid symbol, then the SourceBrowser database may be corrupt. Remove the `.sb` directory and recompile your program with the SourceBrowser option.

If you still get this message, then call SunSoft at 1-800-USA-4SUN or call your local service office. Have the SourceBrowser version ready to give to the dispatcher.

`Source the current class...`

You clicked on the Source button without selecting a symbol to show the source of. By default, ClassBrowser displayed the source of the current class.

Issuing a Command-line Query B

This chapter describes the SourceBrowser command-line environment. This environment is a conventional command-style interface you can access from Sun workstations, as well as from terminals and from workstations emulating terminals.

This chapter is organized into the following sections:

Issuing a Query	page 213
Command-line Options	page 214

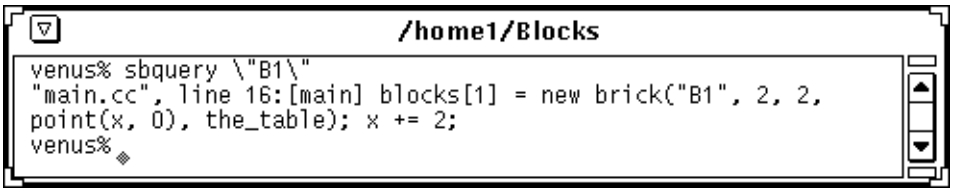
B.1 Issuing a Query

The command-line interface to SourceBrowser is `sbquery`.

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.**

SourceBrowser can search for identifiers, string constants, and search patterns that contain wildcards. When you query for a string constant, you must enclose the string with \" (a backslash followed by quotation marks). The backslash removes any special meaning the quotation mark has in the Shell. In this example, the user queried on the string B1:



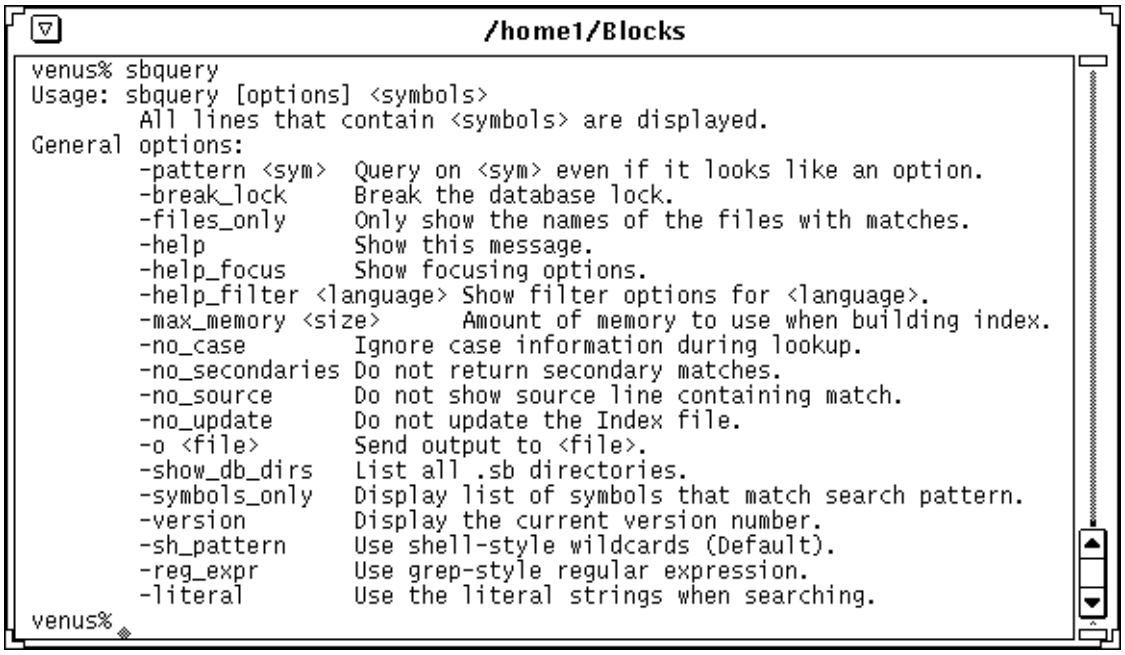
See “Querying for String Constants” on page 153 for more information on searching for a string constant. See “Using Wildcards in Queries” on page 154 for instructions on using wildcards in a query.

By default, SourceBrowser searches for symbols in the database in the current working directory. See Chapter 12, “Working with Multiple Directories," if you want to browse a database stored in another directory.

B.2 Command-line Options

To obtain a list of the SourceBrowser command-line options:

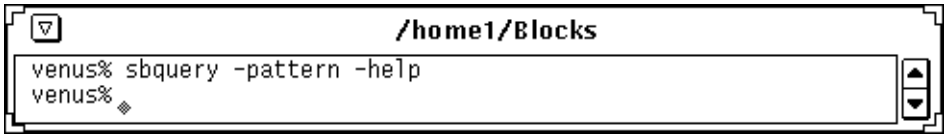
- Type **sbquery** at the SunOS shell prompt.



B.2.1 -pattern <symbol>

The `-pattern` option queries on `symbol`, which may contain special characters, including a leading dash (-). This option allows you to query on a symbol that looks like a command-line option.

In this example, the user queried on the symbol `-help`. `sbquery` distinguishes the symbol `-help` from the command-line option `-help`. `sbquery` returned no matches.

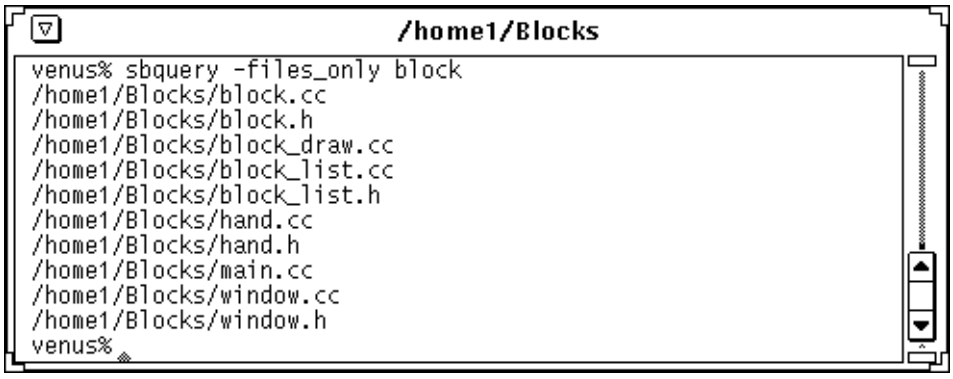


B.2.2 -break_lock

The -break_lock option breaks the lock on a locked database. If the update of the index file is aborted for some reason (for example, a power failure), then SourceBrowser may 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.

B.2.3 -files_only

The -files_only option lists the files in which the symbol you are searching for appears. In this example, the user listed the files in which the symbol block 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, the command

```
edit ‘sbquery -files_only block‘
```

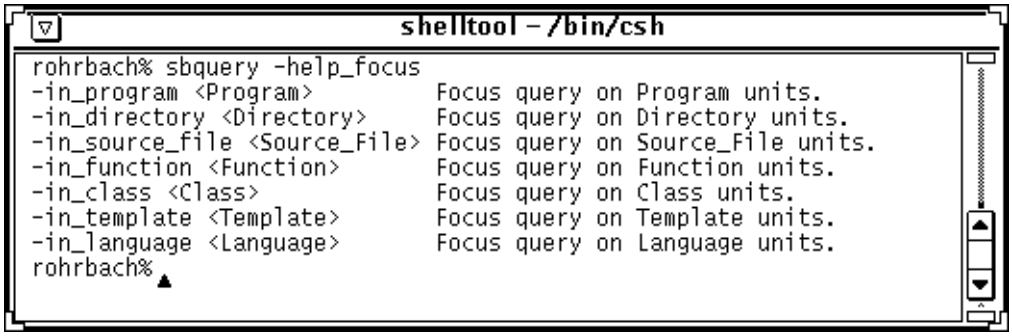
reads into the editor all files containing the symbol block. For information about using back quotes to redirect output to another command, see either the sh(1) or csh(1) man page.

B.2.4 -help

The -help option displays a synopsis of the sbquery command. This is equivalent to entering sbquery without any options.

B.2.5 -help_focus

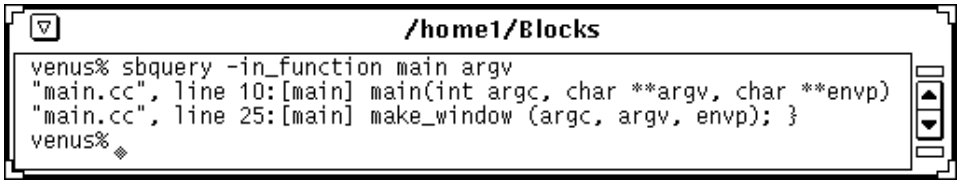
The -help_focus option displays a list of the units of code that you can focus on. You can then issue a focused query; that is, a query limited to specific units of code such as specific programs or functions. Because SourceBrowser automatically searches through all files described by the database in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



To limit your search to a specified unit of code:

- **Type** sbquery **followed by one or more** help_focus **options, followed by the name of the unit of code you want to search in, followed by the symbol you want to search for.**

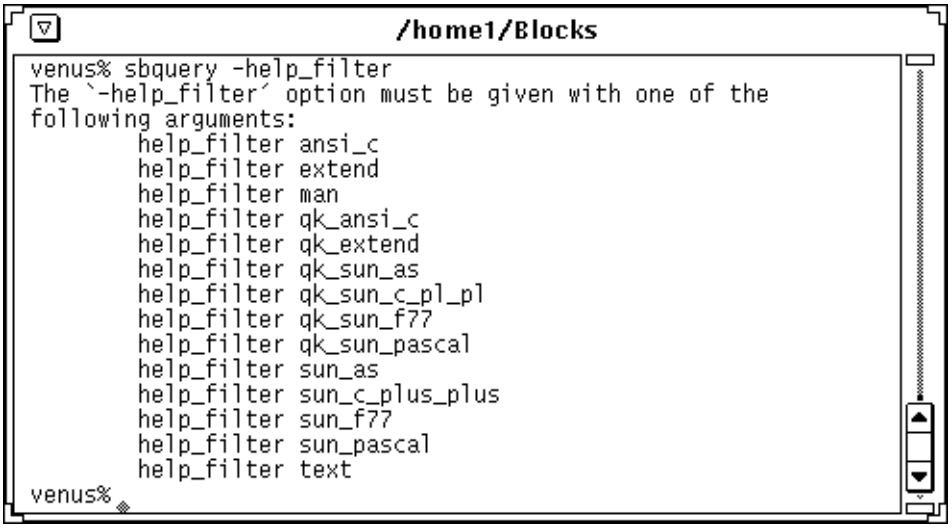
In this example, the user instructed SourceBrowser to find all instances in which argv is used in the function main.



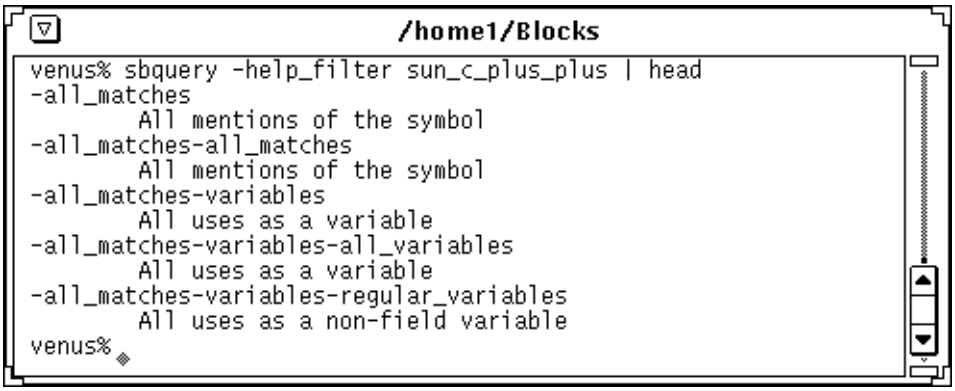
The `-help_focus` option is equivalent to choosing an item from the Focus menu in the SourceBrowser base window.

B.2.6 *-help_filter*

The `-help_filter` option displays a list of the languages for which filtering options are available. Use filtering options to search for symbols based on how they are used in a program.



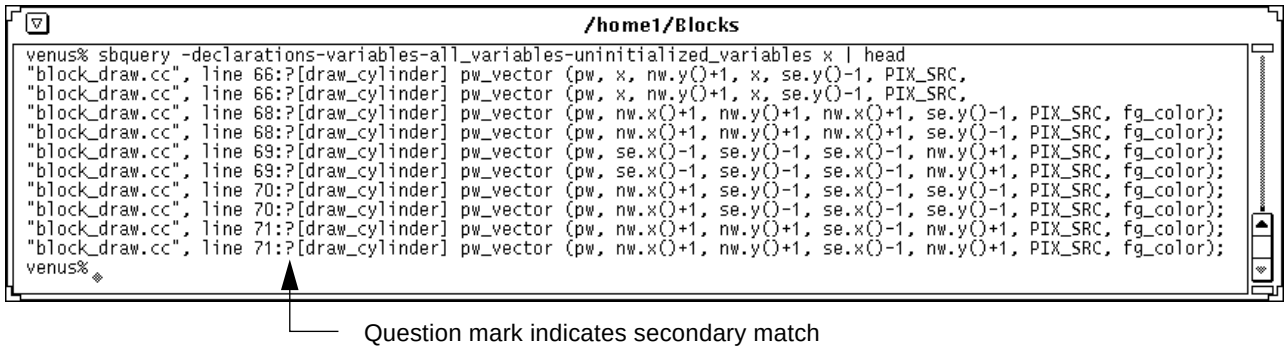
Issuing this option followed by a language supported by SourceBrowser causes SourceBrowser to display a list of filtering options for that language. In this example, the user displayed a partial listing of filtering options for C++.



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

If a question mark (?) appears after the line number, then SourceBrowser found a secondary match. A secondary match is a match inside a macro usage. These matches are called secondary because SourceBrowser is unable to determine precisely how identifiers inside macro definitions are used in a program.

In the following example, the user instructed SourceBrowser to find all instances in which `x` is used as a variable declaration without an initializer. SourceBrowser returned 66 matches, all of which are secondary matches. Following is a partial listing of the output. For more about secondary matches, see Section 7-1, “Issuing a Filtered Query.”



B.2.7 -max_memory <size>

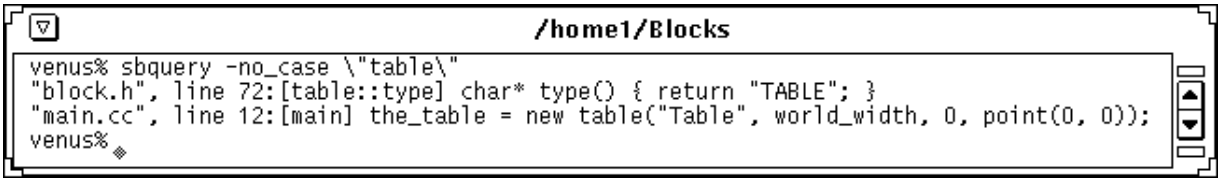
The `-max_memory` option tells SourceBrowser the approximate amount of memory, in megabytes, that it should allocate before it uses temporary files when building the index file.

The `-max_memory` option is equivalent to setting Database Update Allocated in the SourceBrowser Properties window.

B.2.8 -no_case

The `-no_case` option instructs SourceBrowser not to look at case when searching for symbols in the source code that match the query symbol. By default, SourceBrowser makes a match only when the case of the symbol matches the case of the symbol in the source code.

In this example, the user instructed SourceBrowser to ignore all case information and display all matches that contain the string `table`. SourceBrowser returned matches for both *TABLE* and *Table*.



The `-no_case` option is equivalent to setting Case to Ignored in the SourceBrowser Properties window.

B.2.9 -no_secondaries

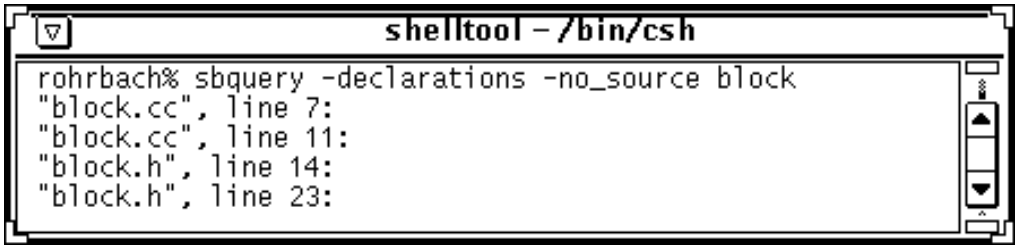
The `-no_secondaries` option instructs SourceBrowser not to return any secondary matches. You might want to turn off secondary matches if you are doing a lot of filtered querying, and the symbols you query are used in a lot of macros. See “Filtering in Macro Definitions” on page 126 for more information on secondary matches.

The `-no_secondaries` option is equivalent to setting Return Secondary Matches to No in the SourceBrowser Properties window.

B.2.10 -no_source

The `-no_source` option causes SourceBrowser to display only the filename and line number associated with each match. If you do not include this option, then SourceBrowser also displays the line of source code containing the match and the name of the function the match was found in, if any.

In this example, the user instructed SourceBrowser to display only the file name and line number of all matches that contain declarations of the symbol `block`.



The `-no_source` option is equivalent to setting Show Match With to Filename and Linenumber Only in the SourceBrowser Properties window.

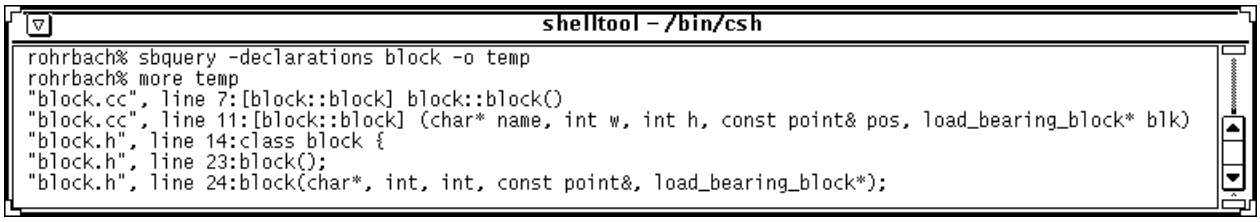
B.2.11 -no_update

The -no_update option 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, then SourceBrowser updates the database and processes your query.

The `-no_update` option is equivalent to setting Database Index to Not Automatically in the SourceBrowser Properties window.

B.2.12 -o <file>

The -o option sends SourceBrowser output to a file rather than to standard output. In this example, the user instructed SourceBrowser to output all declarations that contain the symbol block in the file temp.



B.2.13 -show_db_dirs

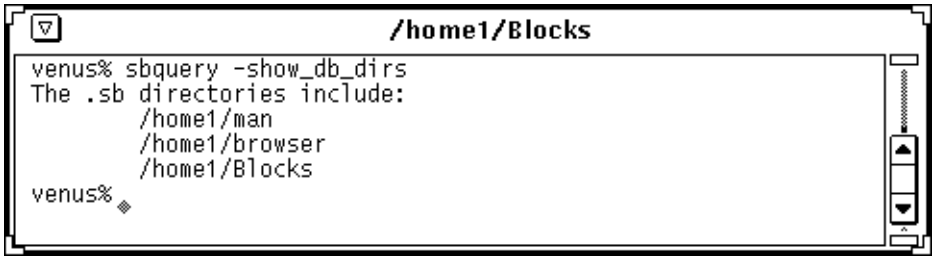
The -show_db_dirs option lists all .sb directories that SourceBrowser scans when you issue a query. The list includes the following:

- The .sb directory in the current working directory
- All other .sb directories you instruct SourceBrowser to read by including either the import or export command in your .sbinit file.

For example, suppose your .sbinit file included the following commands:

```
import /home1/man
export /usr/include into /home1/browser
```

Here is the output when you issue sbquery in the /home/blocks directory with the -show_db_dirs option.

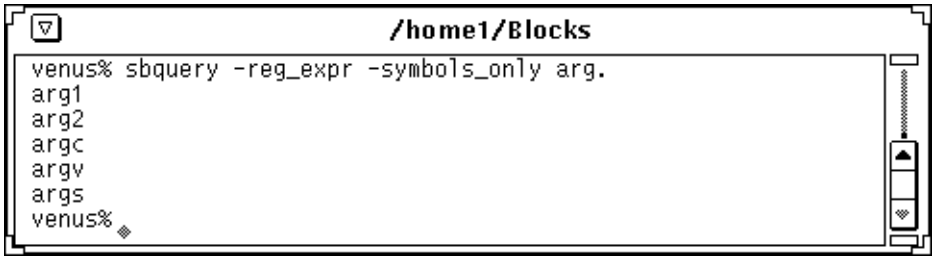


See “Using sbinit to Browse Multiple Directories” on page 182 for more information on the `.sbinit` file.

B.2.14 `-symbols_only`

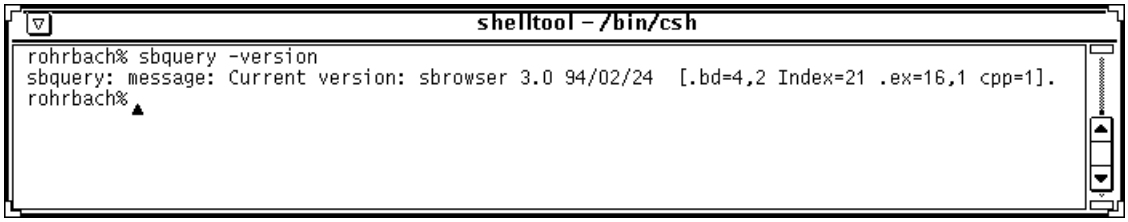
The `-symbols_only` option displays a list of all symbols that match the pattern specified. This option is useful when you use wildcards in a query.

In this example, the user instructed SourceBrowser to display a list of all symbols that match the regular expression `arg.`, where “.” matches any character.



B.2.15 `-version`

The `-version` option displays the current version number of SourceBrowser.

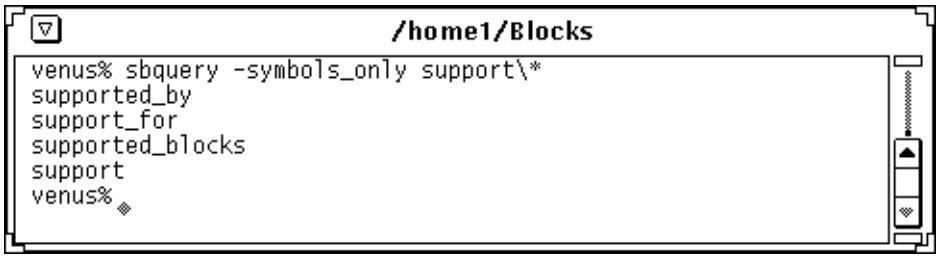


This same information is displayed in the About window.

B.2.16 -sh_pattern

The -sh_pattern option indicates that you are using shell-style expressions when issuing a query that includes wildcards. See sh(1) for more information about shell-style pattern matching. This wildcard setting is the default. You need only include this option if you are doing other pattern matching on the same command line.

In this example, the user instructed SourceBrowser to display a list of all symbols that match the shell pattern support*, where "*" matches any number of characters. Note that the user included a backslash before the asterisk. The backslash removes any special meaning the asterisk has in the Shell.



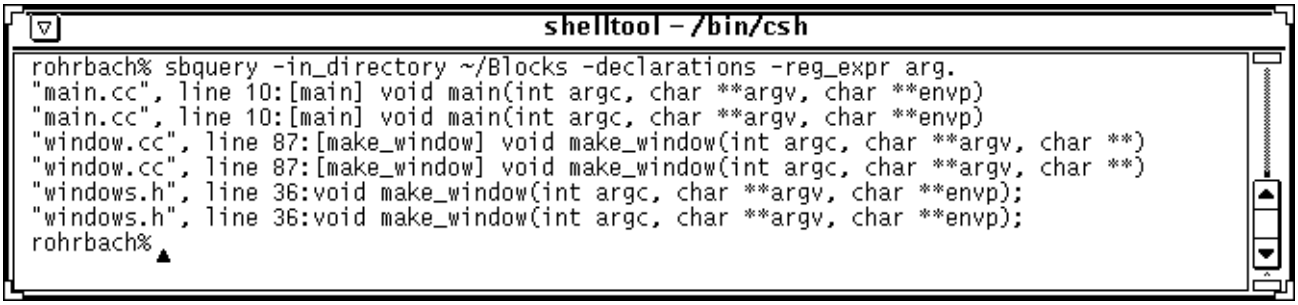
The -sh_pattern option is equivalent to setting Wildcard Style to Shell Style Pattern in the SourceBrowser Properties window.

B.2.17 -reg_expr

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

See “Using Wildcards in Queries” on page 154 for more about using wildcards in queries and about regular expressions.

In this example, the user instructed SourceBrowser to find all declarations of `arg.` where "." matches a single character. The user limited the search to files in `/home1/Blocks` directory.



The main.cc lines are shown twice because there are two matches from argc and argv. The same is true for window.cc and window.h

The `-reg_expr` option is equivalent to setting Wildcard Style to Regular Expression in the SourceBrowser Properties.

B.2.18 `-literal`

The `-literal` option tells SourceBrowser to search for strings that contain the special characters used in shell-style or regular expression patterns. In a literal string query, SourceBrowser ignores any special meaning such characters have.

In this example, the user instructed SourceBrowser to search for all instances of `support*` (and not `supported_by`, `supported_blocks`, or so on). SourceBrowser returned no matches for this query.

The `-literal` option is equivalent to setting Wildcard Style to Literal String in the SourceBrowser Properties window.

Integration with CASE Tools C

C.1 Introduction

The SourceBrowser supports integration with CASE tools. There are two types of integration:

Synchronized Browsing (between SourceBrowser and a CASE tool)

From the SourceBrowser, you can select a symbol in your program and display the design diagram for that symbol in the CASE tool. Conversely, you can select a symbol in a CASE tool, and request that the source code for that symbol be displayed in the SourceBrowser.

Reverse Engineering

Information extracted from the SourceBrowser database is used to create a new design, or update an existing design, in the CASE tool. The protocol was developed jointly with IDE, but can be used to integrate with any CASE tool that supports the protocol described later in this appendix.

This Appendix is organized into the following sections:

CASE Tool Startup	*page 226*
Errors	*page 226*
Synchronized Browsing	*page 227*
Reverse Engineering	*page 232*

C.2 CASE Tool Start Up

In this release of the protocol, the CASE tool and SourceBrowser are not automatically started in order to handle a request. In particular, the SourceBrowser must already be started before the CASE tool can send it an `SPRO_SB_GEN_IDF` or `SPRO_SB_SYNC_IDF` message. Furthermore, the CASE tool must be already started before the SourceBrowser can send it an `SPRO_SB_BROWSE_TOOL` message.

C.3 Errors

In the protocol defined in this Appendix, the ToolTalk error status functionality is used for all error handling. That is, in replying to one of the requests defined below, an error will be registered by filling in the ToolTalk error status code and adding an error status string.

ToolTalk itself will also make use of the error status code for errors and warnings in the control of ToolTalk, for example, when a message is not handled by an application.

Whenever an application (the CASE tool and SourceBrowser) sends a ToolTalk request, it is expected to check the response for an error return. If an error has been returned, the application can then do anything with the error message and code. Presumably, it will print out the error message to the user (for example, in the footer of a window).

The status codes are defined by an **`enum`** (see below) that is used in the error status field of the ToolTalk message. There is an associated error string with each error code, but that can vary at the discretion of the application that "owns" the specific error code.

```
typedef enum sb_idf_status {
```

SB_IDF_OK	= 0	TT_OK
SB_IDF_ERR	= 2047 + 256	TT_ERR_LAST + 256
SB_IDF_ERR_NODIR		Directory not valid.
SB_IDF_ERR_NODB		No database found.
SB_IDF_ERR_NOWRITE		Directory not writable.
SB_IDF_ERR_NODUMP		Failed to geneate IDF dump.

SB_IDF_ERR_NOFILE	Source file not found.
SB_IDF_ERR_INVOBJ	Invalid object type.
SB_IDF_ERR_NOSYMF	Name not found in file.
SB_IDF_ERR_NOSYMD	Name not found in database.
SB_IDF_ERR_INTERN	Internal error.

```
} sb_idf_status;
```

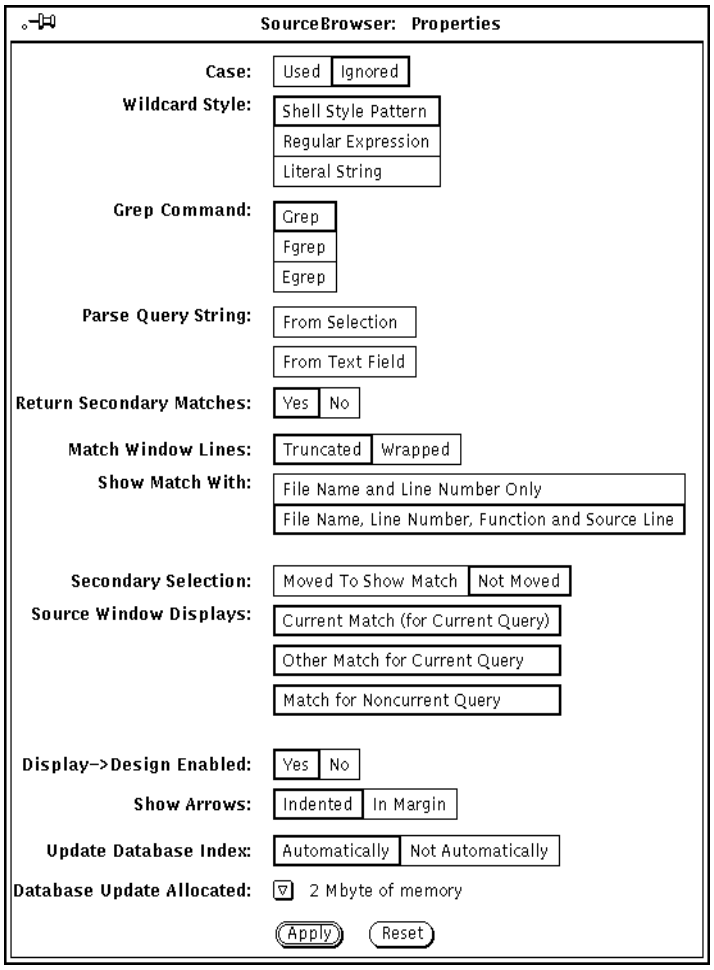
C.4 Synchronized Browsing

The Design menu option in the SourceBrowser Display menu is used for sending a ToolTalk message from the SourceBrowser to the CASE tool. The CASE tool then displays the design diagram encompassing that symbol. Conversely, a design symbol may be chosen in the CASE tool and a ToolTalk message can be sent signaling the SourceBrowser to load the source for that symbol.

The display of the Design menu item is turned OFF by default. To turn it ON: do the following:

- Click **Props...** from the SourceBrowser main window.
- Click **Yes** next to the **Display->Design Enabled** choice item in the popup.
- Click the **Apply** button in the **Properties** sheet. The **Design** menu item is added to the bottom of the **Display** menu.

The **Properties** window is shown on the next page with the **Display->Design Enabled** option set to **Yes**.



The following menu shows the **Design** item that is added to the **Display** window as a result of a **Yes** selection of the **Display->Design Enabled** item in the **Properties** window.

The following menu shows the **Display** window that results from a **No** selection of the **Display->Design Enabled** item in the **Properties** window.

To view a SourceBrowser symbol in the CASE tool, do the following:

- Select a ***symbol*** or ***class::member*** string in the **Source** pane.
- Select the **Design** menu item from the **Display** menu.

C.4.1 SPRO_SB_BROWSE_TOOL *Message*

The SPRO_SB_BROWSE_TOOL ToolTalk message is sent to the CASE tool. This message informs the design diagram editor to display the design for a specified object of a specified type. The message syntax is specified below.

```
(Tt_message
  (Tt_class TT_REQUEST)(Tt_op "SPRO_SB_BROWSE_TOOL")
  (Tt_args (TT_IN "string" directory)
       (TT_IN "string" name)))
```

Tt_args are:

char *directory	The source file is in this directory. This may NOT be in the database directory.
char *object_type	"func", "type", "typedef", "global", "enum", "class, "template", "datamember", "memberfunc", or "libfunc".
char *name	Name of the object. For a "datamember", the name must be in the format class::member..

Errors Returned by the Request:

TT_ERR_NO_MATCH	Message not handled (CASE tool not running).
SB_IDF_ERR_NODIR	Directory not valid (CASE tool cannot open that directory).
SB_IDF_ERR_NODB	No database found (CASE tool has no database in that directory).
SB_IDF_ERR_INVOBJ	object_type is invalid (not one of those listed above).
SB_IDF_ERR_NOSYMD	Name not found in database (either the directory is wrong, is not there, or is incorrectly formatted).

C.4.2 SPRO_Visit_Object *Message*

If the SourceBrowser is running, it responds to the SPRO_Visit_Object ToolTalk request. This message prompts the SourceBrowser to display the source code for the object in the specified file. If the SourceBrowser's current directory is different, it changes its directory. Current queries, graphs, and browsed classes will be lost. The SourceBrowser does a query for the object of the specified type, and if it is found in the specified file, the source code is displayed.

```
(Tt_message
    (Tt_class TT_REQUEST)(Tt_op "SPRO_Visit_Object")
   (Tt_args (TT_IN "string" directory)
        (TT_IN "string" name)
```

```
        (TT_IN "file" file)))
```

`Tt_args` are:

`char *directory`	Directory of the SourceBrowser database
`char object_type`	`"func"`, `"type"`, `"typedef"`, `"global"`, `"enum"`, `"class"`, `"template"`, `"datamember"`, `"memberfunc"`, or `"libfunc"`.
`char *name`	Name of object. For a `"datamember"`, the format of the name must be: `class::member..`
`char *file`	File containing the source for the object.

Errors returned by the request:

`TT_ERR_NO_MATCH`	Message not handled (CASE tool not running).
`SB_IDF_ERR_NODIR`	Directory not valid (CASE tool cannot open that directory).
`SB_IDF_ERR_NODB`	No database found (CASE tool has no database in that directory).
`SB_IDF_ERR_INVOBJ`	`object_type` invalid (not one of those listed above).
`SB_IDF_ERR_NOSYMD`	Name not found in database (either the directory is wrong, is not there, or is incorrectly formatted).
`SB_IDF_ERR_NOSYMF`	Name not found in file (name found, but definition is not in the specified file).

C.4.3 `SPRO_Browse_File` *Message*

This message is sent to request the presentation of a file in the Source pane of the SourceBrowser.

```
(Tt_messageMessage
  (Tt_class TT_REQUEST)(Tt_op "SPRO_Browse_File")
  (Tt_args (TT_IN "string" ignored)
       (TT_IN "string" filename)
       (TT_IN "int" lineno)
```

(TT_IN "int" *ro_rw*)))

`Tt_args` are:

`char *ignored`	Ignored, can be any character string.
`char *filename`	Full pathname of file to load.
`int lineno`	Point SourceBrowser to this line number.
`int ro_rw`	0: make window read only. 1: make window writeable.

Errors returned by the request:

`TT_ERR_NO_MATCH`	Message not handled. (SourceBrowser not running.)
`SB_IDF_ERR_NOFILE`	Source file not found.

C.5 Reverse Engineering

The SourceBrowser and the CASE tool are also capable of sharing data for reverse engineering. Symbols from written code may be shared with the CASE tool in order to generate or regenerate designs.

The SourceBrowser may receive either an `SPRO_SB_GEN_IDF` or `SPRO_SB_SYNC_IDF` request. It then forks the `sbidfdump` utility program to get data from of the SourceBrowser database and write it to a file using the Intermediate Data Format (IDF) file format. (See "*IDF File Format*" for more information on the IDF file.) When `sbidfdump` is finished, it sends an `SPRO_SB_IDF_GENERATED` message to the SourceBrowser including the name of the file or an error code. A reply is then sent to the CASE tool regarding status of the initial request.

C.5.1 `SPRO_SB_GEN_IDF` *Message*

The `SPRO_SB_GEN_IDF` message is a request sent by the CASE tool for SourceBrowser to generate design information in IDF file format for the database in the specified directory. The name of the IDF file is returned. If the lib flag is **0**, then the IDF file will contain information on source files residing in the database directory, otherwise it will contain program information residing in other directories as well as the database directory.

```
(Tt_message
  (Tt_class TT_REQUEST)(Tt_op "SPRO_SB_GEN_IDF")
  (Tt_args (TT_IN "string" directory)
       (TT_IN "boolean" lib)
       (TT_OUT "string" idf_file)))
```

`Tt_args` are:

`char *directory`	Directory of the SourceBrowser database
`int lib`	0: generate data for source files in this directory only.
	1: generate data for source files in other directories, too.
`char *idf_file`	Name of the file containing the resultant data.

Errors returned by the request:

`TT_ERR_NO_MATCH`	Message not handled. (SourceBrowser not running.)
`SB_IDF_ERR_NODIR`	Directory not valid (cannot be opened).
`SB_IDF_ERR_NODB`	No database found (no Sourcebrowser database).
`SB_IDF_ERR_NOWRITE`	Directory not writable (by SourceBrowser).
`SB_IDF_ERR_NODUMP`	Unable to execute `sbidfdump` program.

C.5.2 `SPRO_SB_SYNC_IDF` *Message*

The `SPRO_SB_SYNC_IDF` request asks the SourceBrowser to create an IDF file for the given source file in the given directory. The name of the IDF file that was generated is returned if there is no error.

```
(Tt_message
  (Tt_class TT_REQUEST)(Tt_op "SPRO_SB_SYNC_IDF")
  (Tt_args (TT_IN "string" directory)
       (TT_IN "boolean" lib)
```

```
        (TT_OUT "string" idf_file)
        (TT_IN "file" source_file)))
```

`Tt_args` are:

`char *directory`	Directory of the SourceBrowser database.
`int lib`	This argument is ignored.
`char *idf_file`	Name of the file containing the resultant data.
`char *source_file`	Filename from which to get data.

Errors returned by the request:

`TT_ERR_NO_MATCH`	Message not handled. (SourceBrowser not running.)
`SB_IDF_ERR_NODIR`	Directory not valid (cannot be opened).
`SB_IDF_ERR_NODB`	No database found (no Sourcebrowser database).
`SB_IDF_ERR_NOWRITE`	Directory not writable (by SourceBrowser).
`SB_IDF_ERR_NOFILE`	Source file not found.
`SB_IDF_ERR_NODUMP`	Unable to execute `sbidfdump` program.

C.5.3 `SPRO_SB_IDF_GENERATED` *Message*

The `SPRO_SB_IDF_GENERATED` notice is sent by the `sbidfdump` utility to the SourceBrowser when the IDF file has been generated. If there were no errors, the name of the IDF file is returned. Otherwise, an error code is returned.

```
(Tt_message
  (Tt_class TT_NOTICE)(Tt_op "SPRO_SB_IDF_GENERATED")
  (Tt_args (TT_IN "string" idf_file_name)
       (TT_IN "string" idf_file_type)))
```

`Tt_args` are:

`char *idf_file_name`	Filename of IDF file or error message.
`char *idf_file_type`	"`IDF_FILE`", "`SYNC_FILE`", or "`ERROR`".

Errors returned by the request:

SB_IDF_ERR_NODIR	Directory not valid (cannot be opened).
SB_IDF_ERR_NODB	No database found (no Sourcebrowser database).
SB_IDF_ERR_NOWRITE	Directory not writable (by SourceBrowser).
SB_IDF_ERR_NOFILE	Source file not found.

C.5.4 IDF File Format

The `sbidfdump` utility uses the Intermediate Data Format (IDF) when the SourceBrowser database information is written for reverse engineering with third party CASE tools. The format specification has been adopted from IDE.

An IDF file is easy to read and to parse. The following section describes the syntax of the format and the rules that are followed. There are six major classes of objects defined: functions, library functions, types, enumerations, globals, and typedefs.

The specification for each object of IDF must follow syntactical and structural rules, and may only include these elements:

- Keywords are words that have a special meaning to IDF files. For example, the specification of a function object begins with keyword **`func=,`** and ends with keyword **`endfunc;.`** The complete list of keywords used in IDF can found later in this appendix. All keywords are reserved. They cannot be used to name objects. However, **`func=`** func is a valid statement. In this case, the first **`func`** is part of the keyword, and the second **`func`** is the name of the function object.
- There are six types of identifiers defined in IDF. They are:

  ```
  VERSION_ID
  OBJECT_ID
  TYPE_ID
  SCOPE_ID
  SOURCE_ID
  ARRAY_ID
  ```

The value of VERSION_ID specifies the version of the given IDF file. OBJECT_ID, TYPE_ID, SCOPE_ID, SOURCE_ID, and ARRAY_ID contain information pertaining to different aspects of the given object. Their value is supplied by the programming environment. Detailed descriptions of these identifiers can found later in this appendix.

The conventions for IDF syntax are:

- IDF keywords are always shown in **bold**, lower case letters. White space can be used as a word separator for components of keywords. (For example, **func= & func =** are the same keyword.)
- Identifiers that are supplied by programming environment are always shown in UPPER CASE letters.
- Statements basically begin with a keyword and are followed by an identifier (for example, **func=**test). Each IDF statement requires a newline character (\n) as its terminator. White space can be used as a separator element of statements.
- Braces (**{}**) around statements indicate that there can be zero or more than one occurance of the enclosed statements.
- Brackets (**[]**) around statements indicate that there can be zero or more than one occurrence of the enclosed statements.
- A vertical bar (**|**) between words separates multiple options.
- Angle brackets (**<>**) around text indicate a reference to a subject in the C programming language.

  Each object specification must contain opening and ending statements. (For example, for functions the opening statement is: **func=** OBJECT_ID and the ending statement is: **endfunc;**.) All other statements are optional, and the order in which those statements appear is insignificant. If a specific statement is omitted from the specification, its default value is used.

  Blank lines between statements of a given object specification are ignored during the process.
- Any text following either a **%** or **#** sign is considered a comment, and its content, up to the end of that line, is ignored during the process.

Syntax of the IDF

This section provides detailed information about the IDF file and the six major classes of objects.

IDF Files

intermediate_format_file::=version_spec spec_list

version_spec::=`version`= VERSION_ID *newline*

spec_list::=spec | *spec_list spec*

spec::=function_spec | *libfunc_spec* | *type_spec* | *enum_spec* | *global_spec* | *typedef_spec*

newline::"`\n`*"*

Functions

function_spec::=`func`=OBJECT *newline*

`type`= TYPE_ID *newline*

`scope`=SCOPE_ID *newline*

`source`=SOURCE_ID *newline*

`startline`=INTEGER *newline*

`endline`=INTEGER *newline*

{`parm`=OBJECT_ID [`usage`=FLAG_TYPE] [`scope`=SCOPE_ID] [`arraysize`=ARRAY_ID]`type`=TYPE_ID *newline*}

{`calls`= OBJECT_ID `source`=SOURCE_ID *newline*}

{`calledby`=OBJECT_ID `source`=SOURCE_ID *newline*}

{`globalref`=OBJECT_ID `source`=SOURCE_ID *newline*}

`endfunc`; *newline*

Library Functions

libfunc_spec::=`libfunc`=OBJECT_ID *newline*

type=TYPE_ID *newline*

source=SOURCE_ID *newline*

endlibfunc; *newline*

Types

type_spec::=type=TYPE_ID *newline*

source=SOURCE_ID *newline*

{*field_spec*}

startline=INTEGER *newline*

endline=INTEGER *newline*

endtype; *newline*

field_spec::= field=OBJECT_ID *newline*

level=INTEGER *newline*

type=TYPE_ID *newline*

arraysize=ARRAY_ID *newline*

{*field_spec*}

endfield; *newline*

Enumerations

enum_spec::=enum=OBJECT_ID *newline*

source=SOURCE_ID *newline*

{field=OBJECT_ID value=INTEGER endfield; *newline*}

endenum; *newline*

Globals

global_spec::=global=OBJECT_ID *newline*

type= TYPE_ID *newline*

arraysize=ARRAY_ID *newline*

scope=SCOPE_ID *newline*

source=SOURCE_ID *newline*

startline=INTEGER *newline*

endline=INTEGER *newline*

endglobal; *newline*

Typedefs

typedef_spec::=typedef=OBJECT_ID *newline*

alias=TYPE_ID *newline*

arraysize=ARRAY_ID *newline*

source=SOURCE_ID *newline*

endtypedef; *newline*

Keywords

Table C-1

auto	endtypedef;	source=
alias=	enum=	startline=
arraysize=	extern	type=
calls=	field=	static
calledby=	func=	usage=
endenum;	global=	version=
endfield;	globalref=	d_in | D_IN
endfunc;	level=	d_out | D_OUT
endglobal;	libfunc=	d_in_out | D_IN_OUT
endlibfunc;	parm=	c_in | C_IN
endline=	register	c_out | C_OUT
endtype;	scope=	

Identifiers

VERSION_ID::=`interfile1`

`interfile1` is the current version identifier for IDF. Later versions will be identified by following the same convention with an increased version index number.

OBJECT_ID::= <*C identifier*>

Example, `my_function`

If the `$` sign is used as part of an identifier, it is replace by the letter `S`.

TYPE_ID::= <*C type name*> (also includes **`enum`**)

Example, `struct tm`

ARRAY_ID::= <*C array subscripts*>

Example, `[20][MAXLINE-1]`

If `[]` is specified, it is treated as null.

SCOPE_ID::=`static | extern | auto | register`

The values in this field are not currently enforced: any identifier is accepted. As a result, scopes for functions could be used for purposes other than to indicate the storage class of of the function. For example, SCOPE_ID can specify a real scope (useful in Pascal). For parameters, SCOPE_ID really makes sense if it is a register, although other values could be used.

SOURCE_ID::= <*UNIX filename and path*>

Example, `/usr/lib/libX11.a`

FLAG_TYPE::=`d_in | d_out | d_in_out | c_in | c_out |D_IN | D_OUT | D_IN_OUT |C_IN | C_OUT`

Defaults

The default value for identifier OBJECT_ID is: `__NO_NAME_`*`index`*. The *`index`* starts at zero.

The default value for identifier TYPE_ID is: `int`.

The default value for identifier SCOPE_ID is: `extern`.

There are special default values for SOURCE_ID.

`_UNKNOWN` is used when source information cannot be found.

`_BUILT_IN` is used for C base types.

The default value for identifier FLAG_TYPE is: `D_IN`.

Glossary

This glossary is a collection of terms specific to SourceBrowser. These terms appear throughout the document, and familiarity with their meanings will help you in using SourceBrowser.

active query

A query issued during the current SourceBrowser session. SourceBrowser maintains a history of active queries in the Query menu.

Answerbook

On-line documentation tools that display this manual and all other SPARCworks/PROworks manuals.

Apply button

A button that executes the settings in the Filter, Focus, or Properties windows.

`automount-prefix` command

A command in the `.sbinit` file that instructs SourceBrowser which automounter prefixes to remove from the source file names stored in the database.

back edge

A curved edge in the call graph that represents a recursive cycle. For example, the program may have a function A that calls function B which calls function A.

.bd file

A browser data file created by the compiler when you use the SourceBrowser option during compilation.

browse

To inspect source code for all occurrences of the symbol, string constant, or search pattern that you have specified.

CallGrapher

A SourceBrowser window that provides a graphical view of the function dependencies in a program.

ClassBrowser

A SourceBrowser window that lists the class-defined interfaces and relationships in a C++ program.

ClassGrapher

A SourceBrowser window that provides a graphical view of the class hierarchies in a C++ program.

Class List window

A ClassBrowser pop-up window that lists all classes, structs, unions, and instantiated template classes in a C++ program.

contraction

The process of removing all nodes and edges from the graph that are beyond the contraction radius from the selected nodes. For example, if you set the contraction radius to two, the Grapher removes all nodes and edges at radius three or more from the selected node.

contraction radius

The distance from the selected node(s) beyond which the Grapher removes nodes and edges when you contract the graph.

control area

An area at the top of a window in the OPEN LOOK GUI that contains the buttons and menu buttons.

current class

Current Class: block

The class which ClassBrowser is actively browsing. The current class name is displayed in the Class text field.

current query

The symbol being actively searched for.

edge

A line that connects two nodes in a graph. In the CallGrapher, an edge represents a function call. In the ClassGrapher, an edge represents class inheritance.

expansion

In the call graph, the process of adding nodes that either call or are called by the selected node. In the class graph, the process of adding the base and derived nodes of the selected node.

expansion depth

The distance from the selected nodes within which the Grapher adds nodes and edges to the graph.

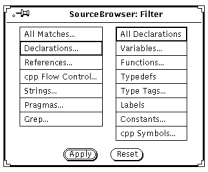
`export` command

A command in the `.sbinit` file that instructs the compilers to write database component files to directories other than the current working directory of the compiler.

filtered query

A query that limits its search for symbols based on the way in which they are used in a program.

Filter window

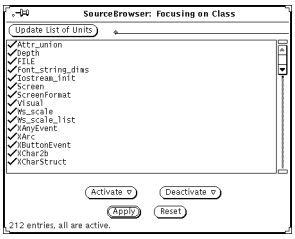


A Sourcebrowser popup window with settings determined by the language(s) of the code you are browsing. Use the settings in this window to limit your search for symbols based on the way in which they are used in a program.

focused query

A query that limits your search to specific items of certain classes of code, such as files or functions.

Focus window



A SourceBrowser popup window that lists all items from the unit of code you selected in the Focus submenu. The Focus window has two menus for activating and deactivating the items you want to use in your search.

identifier

A symbol for which SourceBrowser searches. The identifier can be a variable name, constant name, or user-defined type name.

`import` command

A command in the `.sbinit` file that instructs SourceBrowser to read databases in directories other than the current directory.

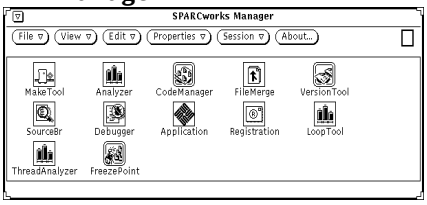
Index file

A file created by SourceBrowser. SourceBrowser uses this file to locate information in the .bd files.

literal string search

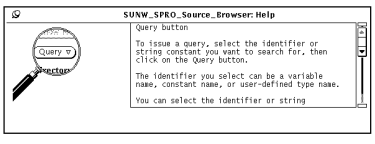
A search for strings or symbols that contain the special characters used in shell-style or regular expression patterns. In a literal string search, SourceBrowser ignores any special meaning such characters otherwise have.

Manager



A desktop tool for managing the SPARCworks/PROworks programming tools. It is a visual organizer that provides easy accessibility for starting, quitting, and managing tools.

Magnify Help



On-line help provided for all SPARCworks/PROworks tools. Magnify Help messages are brief and pertaining to the object under the pointer when you press the Help key.

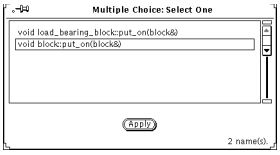
match

A symbol in the code identical to a query. A match is marked by a black arrow in the match and source panes.

match pane

A window area that displays the file name, line number, function, and source code of all matches found by the current query.

Multiple Choices window



A CallGrapher window that pops up when you attempt to graph an overloaded or virtual function in a C++ program. The window lists all instances of the overloaded or virtual function that you can graph.

NewRoot directory

A subdirectory in the `.sb` directory. The `NewRoot` directory stores the `.bd` (browser data) files the compiler creates when you compile your program with the SourceBrowser option. When you issue your first query, SourceBrowser renames this directory `OldRoot`.

node

A name in a graph. In the CallGrapher, a node represents a function. In the ClassGrapher, a node represents a class.

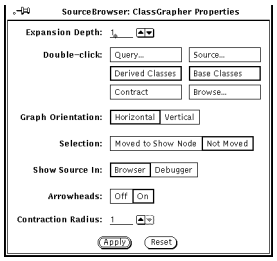
Notice

An OPEN LOOK help facility. Some notices are prompts that inquire about whether you want to continue with a particular action. Other notices are precautionary in that they provide information about the end result of an action.

OldRoot directory

A subdirectory in the `.sb` directory that stores `.bd` (browser data) files. When you compile a source file with the SourceBrowser option, the compiler creates the `.bd` files and stores them in the `NewRoot` directory. When you issue your initial query following a compilation, SourceBrowser renames this directory `OldRoot`.

Properties window



A popup window used to set properties associated with an object, application, or window.

query

An instruction to SourceBrowser to find a specified symbol.

Refd directory

A subdirectory in the `.sb` directory that stores `.bd` (browser data) files for header files.

Reset button

A button in the Filter, Focus, and Properties windows that restores the window controls to their previous state.

`.sb` subdirectory

The SourceBrowser subdirectory that stores the `.bd` files and the index file. Unless you specify otherwise, the `.sb` directory is created in the current working directory from which the compiler is called and read from the current working directory of SourceBrowser.

.SBdefaults file

A file in the user's home directory that contains the SourceBrowser, CallGrapher, ClassGrapher, and ClassBrowser default property settings. All SourceBrowser sessions use the default settings in this file.

`.sbinit` file

An optional text file used by SourceBrowser and the compilers to read from and write to a database in a directory other than the one in the current working directory.

sbquery

The command-line interface to SourceBrowser.

sbrowse

The Open Windows interface to SourceBrowser.

sbtags

Creates tags files used when scanning source files (see the sbtags man page).

Scanning

A quick and convenient method for obtaining browsing inforamtion about source files that do not compile completely.

secondary match

A match in which SourceBrowser finds the identifier inside a macro definition or argument list of a macro expansion. Secondary matches are identified by a question mark in the arrow and only appear when you issue a filtered query.

SourceBrowser option

The option that causes the compiler to add information to the SourceBrowser database. The option is either `-xsb`, `-sb`, or `-b`, depending on the language your program is written in. **REMINDER:** The -xsb option is not available for systems running SunOS 4.1.X.

SUNPRO_SB_EX_FILE_NAME

The environment variable that tells SourceBrowser where to find the `sun_source_browser.ex` file. You must set this variable if `sun_source_browser.ex` is installed in a nonstandard location.

`sun_source_browser.ex` file

The configuration file for SourceBrowser.

SUNPRO_SB_INIT_FILE_NAME

The environment variable that tells SourceBrowser where to search for the `.sbinit` file. The SourceBrowser default is to look in the current working directory.

wildcard query

A search pattern that includes shell-style patterns, regular expressions, or literal strings.

Index

Symbols

.bd files, 11-171 to ??
.sbinit file, 12-182 to 12-188

A

About button, 1-7
active queries defined, 4-59
Add Node button, 5-76
ANSI C programs, 1-4
AnswerBook, xvi
Arrowheads property, 5-92
automounted path, 12-187

B

back edge, 5-63
bd files, 11-171 to ??
break_lock option, B-216
Browse button
    in ClassBrowser window, 6-98
    in ClassGrapher window, 5-75, 6-96
Browse menu in Class List window
    In Current Window, 6-95
    In New Window, 6-99
Browse menu in ClassBrowser
    In Current Window, 6-98
    In New Window, 6-99
    List All Classes, 6-95
browsing
    classes from ClassGrapher, 5-75
    from multiple machines, 12-186
    in an automounted path, 12-187

C

C++ programs, 1-4
CallGrapher, 5-62 to 5-92
CallGrapher Properties window, 5-85 to 5-92
    Arrowheads, 5-92
    Contraction Radius, 5-92
    Expansion Depth, 5-88
    Function Names, 5-91
    Graph Orientation, 5-90
    Selection, 5-91
    Show Source In, 5-92
case in queries for sbquery, B-220
Case property, 10-163
class
    base, 6-96, 6-104
    current, 6-95
    derived, 6-96, 6-104
    friend, 6-96, 6-104
Class List window, 6-95, 6-107 to 6-110

ClassBrowse button, 6-94
ClassBrowser, 6-93 to 6-116
activating from ClassGrapher, 6-96
activating from SourceBrowser, 6-94
Base classes menu, 6-96, 6-104
Class List window, 6-107 to 6-110
Class menu, 6-105
cloning, 6-98
Data Members display pane, 6-96
Derived classes menu, 6-96, 6-104
Friend classes menu, 6-96, 6-104
Function Members display pane, 6-96
Graph menu, 6-106
how to quit, 6-117
inherited members, 6-102
private, protected, and public class members, 6-102
Properties window, 6-114 to 6-116
Query button, 6-110
Source button, 6-112
Text field, 6-95
ClassBrowser Properties window
Display Defaults, 6-103, 6-115
Graph Mode, 6-116
Maximum History, 6-116
Show Glyphs, 6-115
Show Source In, 6-116
ClassGrapher, 5-70 to 5-92
ClassGrapher Properties window, 5-85 to 5-92
Arrowheads, 5-92
Contraction Radius, 5-92
Expansion Depth, 5-88
Graph Orientation, 5-90
Selection, 5-91
Show Source In, 5-92
command-line
issuing queries from, B-213
viewing options, B-214
command-line options
break_lock, B-216, B-226
files_only, B-216
help, B-217
help_filter, B-218
help_focus, B-217
max_memory, B-220
no_case, B-220
no_secondaries, B-221
no_source, B-221
no_update, B-222
o, B-223
pattern, B-215
reg_expr, B-225
sh_pattern, B-225
show_db_dirs, B-223
symbols_only, B-224
version, B-224
compiler options, 2-29
compiling
for SourceBrowser, ?? to 2-31
in an automounted path, 12-187
Contraction Radius property, 5-92
current class, 6-95
current directory, changing, 2-36
current match, 3-43
current query, 3-43

D

data members, 6-96
database
contents, 11-171
locked, 3-49, 13-190, B-216
updating, B-222
Database Update Allocated property, 10-167
Debugger
syncing with SourceBrowser, 8-141 to 8-147
using with ClassBrowser, 8-147 to 8-150
directories, changing, 2-36
disk space, saving, 12-184
Display Defaults property, 6-115

E

edge
as defined in CallGrapher, 5-63

as defined in ClassGrapher, 5-71
Edit menu in Grapher, 5-79
Edit menu in SourceBrowser
Disable Editing, 8-141
Enable Editing, 8-140
Remove All Queries, 4-60
Remove Current Match, 4-60
editing source code, 8-139 to 8-141
editing the graph, 5-79
editor, 1-22
environment variables
HELPPATH, 2-38
SUNPRO_SB_ATTEMPTS_MAX, 13-190
Expand menu
in CallGrapher, 5-68
in ClassGrapher, 5-72
expand operations
in CallGrapher, 5-67
in ClassGrapher, 5-72
Expansion Depth property, 5-88
export command, 12-184 to 12-186

F

files_only option, B-216
Filter menu
Set Filter, 7-120
Filter window, 7-120
Apply button, 7-121
Reset button, 7-123
filtering a query, 7-119 to 7-129
disabling the filter, 7-124
enabling the filter, 7-120
filtering in macro definitions, 7-126
obtaining very specific matches, 7-128
reducing the number of matches, 7-124
resetting the filter, 7-123
finding a node, 5-78
Focus menu
Clear Focus, 7-136
Set Focus, 7-130
Focus window
Activate menu, 7-131
Apply button, 7-132
Deactivate menu, 7-131
Reset button, 7-135
Update List of Units button, 7-135
focusing a query, 7-129 to 7-137
focusing on more than one unit of code, 7-137
resetting the focus, 7-135
setting the focus, 7-130
updating the Focus window, 7-135
FORTRAN programs, 1-4
function
inline, 6-103
virtual, 6-96, 6-103
function members, 6-96
Function Names property, 5-91

G

Graph menu
CallGraph, 5-62
ClassGraph, 5-70
Graph Mode property, 6-116
Graph Orientation property, 5-90
graphing class inheritance, 5-70 to 5-92
adding a node, 5-76
browsing classes, 5-75
editing the graph, 5-79
expanding classes, 5-72
finding a node, 5-78
from ClassBrowser, 6-106 to 6-107
redoing the graph layout, 5-81
graphing function relationships, 5-62 to 5-92
adding a node, 5-76
editing the graph, 5-79
expanding functions, 5-67
finding a node, 5-78
redoing the graph layout, 5-81

H

hash value in .bd files, 11-173, 11-174

Help Facilities
answerbook, xvi
help option, B-217
help_focus option, B-217
HELPPATH environment variable, 2-38

I

identifiers, searching for, 3-42
include files, 12-184
index file, 11-171 to 11-177
inherited class members, 6-101
inline function, 6-96, 6-103
issuing a query from ClassBrowser, 6-110
issuing a query from the Grapher, 5-81

L

linking and .bd files, 11-177
literal option, B-226
locked database, 3-49, 13-190, B-216

M

Magnify Help, xvi
makefile
rebuild database using, 2-28
match, 3-43, 4-51
defined, 1-4, 1-6
how to remove, 4-59
next, 4-52
previous, 4-54
secondary, 7-126
match pane, 3-44
Match Window Lines property, 10-165
max_memory option, B-220
Maximum History property, 6-116
memory allocation, B-220
multiple ClassBrowsers, 6-98

N

Next Match
Any Query, 4-56
Directory, 4-53
File, 4-53
Function, 4-53
next match defined, 4-52
no_case option, B-220
no_secondaries option, B-221
no_source option, B-221
no_update option, B-222
node
adding to the graph, 5-76
as defined in CallGrapher, 5-63
as defined in ClassGrapher, 5-71
finding in the graph, 5-78
removing from graph, 5-79

O

o option, B-223
On-Line Help
answerbook, xvi
overloaded function, 5-64

P

Parse Query String property, 10-164
Pascal programs, 1-4
pattern option, B-215
Prev Match
Any Query, 4-56
Directory, 4-55
File, 4-55
Function, 4-55
previous match, 4-54
previous query, 3-47
private, protected, and public class members, 6-101
properties in Graphers, 5-85 to 5-92

Q

query
defined, 3-41
how to highlight, 3-48
how to ignore case, 10-163

how to include command-line options, 10-164
how to include wildcards, 10-163
how to issue, 3-41 to 3-47
how to issue from sbquery, B-213
how to remove, 4-60
how to search for C++ operator functions, 3-48
how to speed up your search, 3-48
setting a filter, 7-119 to 7-129
setting a focus, 7-129 to 7-137
tips on issuing, 3-47
viewing previous queries, 3-47
what to do if your query fails, 3-49
Query button
in CallGrapher and ClassGrapher, 5-82
in ClassBrowser, 6-110
query symbol flag, 1-22
querying for classes, 6-110 to ??

R

redisplaying the current match, 4-51
reg_expr option, B-225
regular expressions, 9-154, B-225
removing matches, 4-59
removing queries, 4-60
Return Secondary Matches property, 7-127, 10-164

S

sb compiler option, 2-29
SB_ATTEMPTS_MAX environment variable, 13-190
`sbbuild`, ?? to 2-31
sbinit file, 12-182 to 12-188
export command, 12-184 to 12-186
sbquery, B-213
-break_lock option, B-216
-files_only option, B-216
-help option, B-217
-help_filter option, B-218
-help_focus option, B-217
-literal option, B-226
-max_memory option, B-220
-no_case option, B-220
-no_secondaries option, B-221
-no_source option, B-221
-no_update option, B-222
-o option, B-223
-pattern option, B-215
-reg_expr option, B-225
-sh_pattern option, B-225
-show_db_dirs option, B-223
-symbols_only option, B-224
-version option, B-224
sbrowser, 2-35
sbtags, 2-30
Scan, 2-30
SCCS, 8-140
searching
for string constants, 9-153
using wildcards, 9-154
secondary match, 7-126
selection handling, 10-166
Selection property
in CallGrapher and ClassGrapher, 5-91
in SourceBrowser, 10-165
sh_pattern option, B-225
shell-style expressions, 9-154, B-225
Show Arrows property, 10-166
Show Glyphs property, 6-115
Show Match With property, 10-165
Show Source In property
in CallGrapher and ClassGrapher, 5-92
in ClassBrowser, 6-116
show_db_dirs option, B-223
Solaris, xii
Source button
in CallGrapher and ClassGrapher, 5-84
in ClassBrowser, 6-112
source code

displaying, B-221
truncating lines, 10-165
Source Window Displays property, 10-166
SourceBrowser
database, see database
starting, 2-35
SourceBrowser Properties window, 10-162 to 10-167
Case, 10-163
Database Update Allocated, 10-167
Match Window Lines, 10-165
Parse Query String, 10-164
Return Secondary Matches, 10-164
Selection, 10-165
Show Arrows, 10-166
Show Match With, 10-165
Source Window Displays, 10-166
Update Database Index, 10-167
Wildcard Style, 10-163
string constants, searching for, 9-153
SunOS 4.1.X, xiii
SunOS 5.0, xii, xiii
SUNPRO_SB_ATTEMPTS_MAX environment variable, 13-190
symbolic link, 12-179 to 12-180
symbols_only option, B-224
System V Release 4 (SVR4), xiii

U

UCB BSD 4.3, xiii
Update Database Index property, 10-167
updating the Focus window, 7-135

V

version option, B-224
View menu in Graphers
Find Node, 5-78
Redo Layout, 5-81
virtual function, 6-96, 6-103

W

Wildcard Style property, 10-163
wildcards, 9-154, B-225