CASEVision™/WorkShop MegaDev User's Guide

Document Number 007-2114-002

CONTRIBUTORS

Written by Douglas B. O'Morain and Carol Geary
Illustrated by Douglas B. O'Morain and Carol Geary
Edited by Christina Cary
Production by Gloria Ackley
Engineering contributions by David Henke, Stuart Liroff, Ashok Mouli, Michey Mehta, Roy Mittendorff, Anil Pal, Andrew Palay, Jack Repenning, Krishna Sethuraman, Ravi Shankar, and Shankar Unni

CASEVision™ / WorkShop MegaDev User's Guide
Document Number 007-2114-002

Contents

Figures

Tables

Introduction

This guide describes the Fix and Continue and Browser utilities. These tools are part of CASEVision™/WorkShop, a suite of graphical, interactive, computer-aided software engineering (CASE) tools designed especially for programmers who develop and maintain C and C++ libraries and applications.

What This Guide Contains

This guide describes the Fix and Continue feature of the Debugger and the Browser. Chapters 2-4 cover Fix and Continue, and Chapters 5-7 cover the Browser. The guide contains the following chapters:

- Chapter 1, “CASEVision/WorkShop MegaDev,” gives an overview and tells where to find information about each application.
- Chapter 2, “Getting Started with Fix and Continue,” tells you how to install Fix and Continue and run it on your C or C++ files that have been compiled with debug information. This chapter also provides a brief overview of the Fix and Continue utility’s user interface and explains some basic concepts.
- Chapter 3, “Using Fix and Continue: A Sample Session,” helps you learn how to perform the basic tasks that the Fix and Continue utility allows, such as making changes to functions and running the program with compiling or linking. Each task description is accompanied by a corresponding tutorial session.
- Chapter 4, “Fix and Continue Reference,” contains a complete description of the Fix and Continue utility’s graphical user interface.
- Chapter 5, “Getting Started with the C++ Browser,” tells you how to install and use the C++ Browser on your C++ source or library files. This chapter also provides an overview of the C++ Browser’s user

interface and explains some important concepts you'll need to know to get the most out of the C++ Browser.

- Chapter 6, "Using the C++ Browser: A Sample Session," helps you learn how to perform the basic tasks the C++ Browser allows, such as examining class structures, making queries on C++ objects and methods, and viewing class and calling structures. Each task description is accompanied by a corresponding tutorial session.
- Chapter 7, "C++ Browser Reference," contains a complete description of the Browser's graphical user interface. It tells how to customize some aspects of the interface.

The glossary defines key terms for both Fix and Continue and the Browser.

What You Should Know Before Reading This Guide

This guide assumes that you're familiar with C, C++, and object-oriented programming, and have had some experience with the CASEVision / WorkShop tools, particularly the Static Analyzer and Debugger.

Related Information

Fix and Continue and the Browser are layered on the core CASEVision / WorkShop toolset (available from Silicon Graphics, Inc.). For further information about related tools, refer to the following documents:

- *CASEVision/WorkShop User's Guide*, which contains detailed information on how to use the CASEVision tools: the static analyzer, the debugger, the performance analyzer, and the build manager.
- *C++ Programmer's Guide*, which describes the Silicon Graphics C++ programming environment.
- *IRIS ViewKit User's Guide*, which describes how to create programs using IRIS ViewKit, a C++ toolkit that provides commonly needed facilities for applications based on the IRIS user interface toolkit.
- *CASEVision Environment Guide*, which contains general information on using the CASEVision environment, the Silicon Graphics computer-aided software engineering (CASE) tools.

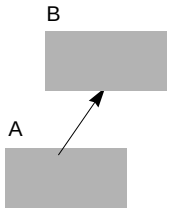
- *MIPSpro Compiling, Debugging and Performance Tuning*, which discusses how to compile, debug, and tune the performance of programs written in the Silicon Graphics development environment (C, Fortran, and C++).

In addition, the following manuals provide information about an earlier implementation of the C++ language from Silicon Graphics (that is, not Delta / C++).

- *C++ Language System Overview*, which contains an overview of new language features of C++. Most of the extensions take the form of removing restrictions on what can be expressed in C++.
- *C++ Language System Product Reference Manual*, which contains a general description of the C++ language.
- *C++ Language System Library*, which discusses the iostream support in the C++ library and describes a data-type complex that provides the basic facilities for using complex arithmetic in C++.

Conventions

Below are the typographical and graphic conventions used in this guide:

- **Bold**—Functions, option flags, and classes.
- *Italics*—Filenames, button names, field names, variables, emphasis, glossary terms, and IRIX commands.
- Regular—Menu and window names, data types, keywords, and text.
- “Quoted”—Menu choices.
- `Fixed-width`—Code examples and command syntax.
- **`Bold fixed-width`**—User input. Nonprinting **`<keys>`** are bracketed.
- 

Graphic convention—Pull-down or popup menus.

Chapter 1

CASEVision / WorkShop MegaDev

The CASEVision / WorkShop MegaDev software is layered on the core CASEVision / WorkShop toolset. MegaDev provides more advanced tools for the development of applications, featuring

- Fix and Continue
- C++ Browser

About WorkShop MegaDev Tools

The programming language C++ offers a good balance between flexibility and performance. Programmers accept C++ for its portability and use of libraries. Object-oriented programming in C++ promises efficient execution, structured classification, and substantial reuse of code. Nevertheless, you might encounter some of these problems as you build a large software system in C++:

- expensive recompilations whenever a class definition changes
- a long edit-compile-debug cycle for minor changes to code
- an inability to support shared C++ libraries
- a lack of high-level libraries
- difficulty understanding code because key information is hidden in class hierarchies

As a solution to these problems, the WorkShop Pro C++ package speeds compiles, improves debugging, simplifies user interface design, and helps you to visualize complex class relationships. Thus it enables you to be a more productive C++ programmer. Here's a quick description of what each tool allows you to do:

Fix and Continue
: Redefine existing C or C++ functions, and then continue your execution without recompiling your entire code set.

C++ Browser
: Display information about classes and their relationships as query results or class / call graphs.

The following sections provide more detailed descriptions of each tool included in the MegaDev package.

Fix and Continue

Fix and Continue lets you redefine existing C or C++ function definitions in several ways, then disable or re-enable individual redefinitions. You continue running your program in the Debugger without recompiling the entire system or linking. You can examine the differences between the original and redefined functions. Fix and Continue accepts input from the Debugger command line as an alternative to using Debugger menus.

Fix and Continue is integrated with the WorkShop 2.4 Debugger, so you can set breakpoints inside the redefined code. Special notation for line numbers in the Debugger (and in many of its views) shows when a given function is redefined, in addition to unique color highlighting of the source code.

Fix and Continue is covered in Part I of this guide and introduced in Chapter 2, “Getting Started with Fix and Continue.”

C++ Browser

The C++ Browser lets you view the structure of any set of C++ classes. It provides a global, graphical view of interclass relationships such as inheritance, containment, and interaction within a set of classes. The convenient, outline-based display shows class relationships and internal class structure from the point of view of any class, member, or method in the hierarchy. A history facility lets you quickly shift among classes. Complete source listings of any class or member are available at the click of a button. The C++ Browser uses a compiler-generated static analysis database to answer your queries.

The C++ Browser is part of the WorkShop 2.4 Static Analyzer, so you can use it as an interactive development and maintenance aid for application source code as well as a reference tool for exploring the structure of C++ programs or libraries.

The C++ Browser is covered in Part II of this guide and introduced in Chapter 5, “Getting Started with the C++ Browser.”

Setting Up Your System

CASEVision / WorkShop MegaDev requires installation of IRIX™ system software version 5.3 or greater, ToolTalk 1.1 or greater, and the Integrated Development Option (IDO).

The C++ Browser requires the WorkShop 2.4 Execution Environment, CASEVision 2.4, and CASEVision / WorkShop 2.4.

To determine what software is installed on your system, enter the following at the shell prompt:

```
% versions
```

If the software items mentioned in this section are not installed, consult your sales representative, your service provider, or (in the U.S.) call the Silicon Graphics Technical Assistance Center at 1-(800)-800-4SGI. To order additional memory, consult your sales representative, or call 1-(800)-800-SGI1. Table 1-1 shows the base product needed to run each module in WorkShop Pro C++.

Table 1-1 WorkShop Pro C++ Special Dependencies

Element	Requirement
Fix and Continue	CASEVision / WorkShop Debugger
C++ Browser	CASEVision / WorkShop Static Analyzer

If you have all the software and memory you need, you're ready to install the CASEVision / WorkShop Pro C++ software. Consult the *IRIS Software Installation Guide* for general instructions on software installation. For specific installation instructions, see the *CASEVision/WorkShop MegaDev Browser Release Notes*.

Chapter 2

Getting Started with Fix and Continue

This chapter shows you how to start the Fix and Continue utility, explains pertinent concepts, and presents an overview of its command-line and graphical user interface.

Note: In Chapters 2 through 4, the word *function* means a C function or a C++ member or nonmember function.

This chapter contains the following sections:

- "Starting Fix and Continue"
- "Redefining Functions Using Fix and Continue"
- "The Fix and Continue Environment"

Starting Fix and Continue

Fix and Continue is integrated with the CASEVision / WorkShop Debugger, *cvd*. To start the Debugger from a shell command line, use the following syntax:

cvd [-pid *pid*] [-host *host*] [*executable* [*corefile*]] [&]

You can specify the executable from the Debugger and by default the host is local, so all you really need to enter is **cvd**.

You issue Fix and Continue commands graphically from the Fix+Continue pulldown menu of the Debugger main window. You may also issue Fix and Continue commands from the Debugger command line (cvd>).

Redefining Functions Using Fix and Continue

Fix and Continue gives you the ability to make changes to a program you are debugging without having to recompile and link the entire program, and then continue debugging the code. With Fix and Continue, you can edit a function, parse the new function, and continue execution of the program being debugged. Fix and Continue enables you to speed up your development cycle significantly.

Table 2-1 compares the cycle time in seconds between a full rebuild and a Fix and Continue for three typical programs.

Table 2-1 Fix and Continue Compile Time Cycle

Example	Time to Rebuild	Time to Fix+Continue
Program A	0:06	0:02
Program B	0:33	0:06
Program C	5:24	0:49

Fix and Continue Functionality

Fix and Continue lets you:

- Redefine existing function definitions
- Disable, re-enable, save, and delete redefinitions
- Set breakpoints in and single-step within redefined code
- View the status of changes
- Examine differences between original and redefined functions

The basic cycle of using Fix and Continue is shown in Figure 2-1.

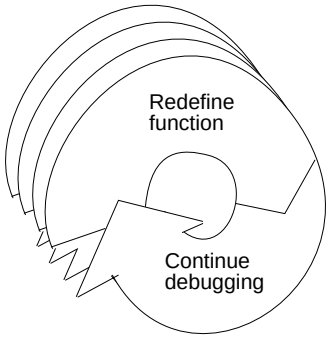


Figure 2-1 Fix and Continue Cycle

A typical session would be the following:

1. Using the Fix and Continue commands, you redefine a function. When you continue executing the program, the Debugger attempts to call the redefined function. If it cannot, an information popup appears, and the redefined function will be executed the next time the program calls that function.
2. You redefine other functions, alternating between debugging, disabling, re-enabling, and deleting redefinitions. You might save function redefinitions to their own files, or save files to a different name, to be used later with the present or with other programs.

Frequently during debugging you can review the status of changes by listing them, showing specific changes, or looking at the Fix and Continue Status View. You can compare changes to an individual function or to an entire file with the compiled versions. When satisfied with the behavior of your application, you save the file, replacing the compiled source.

Fix and Continue/WorkShop Integration

Using Fix and Continue affects these WorkShop tools:

- The WorkShop Debugger Main View, the Source View, and the Fix and Continue Status window make a clear distinction between compiled and redefined code, and allow editing only in redefined code.
- The different WorkShop views that are knowledgeable about redefined code:
  - Call Stack
  - Trap Manager
  - Debugger command-line

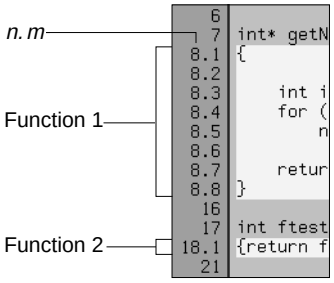


Figure 2-2 Line Numbers in Decimal Notation

How Redefined Code Is Distinguished From Compiled Code

Redefined functions have an identification number and special line numbers, and in the Debugger views, are color-coded.

Line numbers in the compiled file stay the same, no matter how redefined functions change. However, when you begin editing a function, the line numbers of the function body are represented in decimal notation (*n*.1, *n*.2, ..., *n*.*m*). *n* is the compiled line number where the function body begins. *m* is the line number relative to the beginning of the function body, starting with 1.

Figure 2-2 shows two redefined functions. Function 1 replaces lines 8-15. Function 2 replaces lines 18-20. Although its three lines are now one, the following line number is still 21.

The Call Stack and Trap Manager both use function-relative decimal notation when referring to a line number within the body of a redefined function.

The Debugger command line reports ongoing status. In addition to providing the same commands available from the menu, edit commands allow you to add, replace, or delete lines from files. You can easily operate on several files at once.

Restrictions on Fix and Continue

Fix and Continue has the following restrictions when you fix a function in which you have stopped:

- You may not add, delete, or reorder the local variables in a function.
- You may not change the type of a local variable.
- You may not change a local variable to be a register variable, and vice-versa.
- You may not add any function calls that increase the size of the parameter area.
- You may not add an **alloca** function to a frame that did not previously use an **alloca**.
- Both the old and new functions must be compiled with *-g*.

  In other words, the layout of the stack frames of both the old and new functions must be identical for you to continue execution in the function that is being modified. If not, execution of the old function continues, and the new function is executed the next time it is called.
- If you redefine functions which are in but not on top of the call stack, the modified code will not be executed when they combine. Modified functions will be executed only on their next call or on a re-run.

  For example, consider the following call stack:

  ```
  foo()
  bar()
  foobar()
  main()
  ```

  1. If you redefine **foo()**, you can continue execution provided the layout of the stack frames are same.
  2. If you redefine **bar()** [or **foobar()**], the new code will not be executed when **foo()** returns. The code will be executed only on the next call of **bar()** [or **foobar()**].
  3. If you redefine **main()** after you have run, it will be executed only when you re-run.

The Fix and Continue Environment

The interface to Fix and Continue is through the Fix+Continue menu and its associated windows: Status, Message, and Build Environment. These windows are completely dependent on Fix and Continue, and do not operate unless it is installed.

For more complete information on all of the Fix and Continue menus, windows, and functions, see Chapter 4, “Fix and Continue Reference.”

Debugger With Fix and Continue Support

Without Fix and Continue, the Debugger source views are `Read-Only` by default. That is so you can examine your files with no risk of changing them. When you select “Edit” from the Fix+Continue menu, the Debugger source code status indicator (in the lower-right corner of the Debugger window—see Figure 4-20) remains `Read-Only`. Fix and Continue edits are saved in an intermediate state and must be explicitly written with the “Save File+Fixes As...” option to be saved.

When you edit a function, it is color-highlighted. Then, if you switch to the compiled version, the color changes to show that there is a redefinition. If you try to edit the compiled version, the Debugger beeps, indicating it is `Read-Only`.

When you have completed your edits and wish to see the results, click “Parse and Load.” When the Parse and Load has executed successfully, the color changes again. If the color doesn’t change, there may be errors, and you should check your “Message Window...” view.

GUI Debugger Command Line

Just as you can enter any *dbx* command at the Debugger command line, you can enter Fix and Continue commands there.

Change ID, Build Path, and Other Concepts

The Fix and Continue methods for accessing functions through ID numbers, finding files, and so forth, are discussed below.

Identifying Functions

Each redefined function is numbered with a change ID. Its status is redefined, enabled, disabled, deleted, or detached.

Finding Files

Fix and Continue needs to know where to find include files and other parameters specified by compiler build flags (compiler options). You can set the build environment for all files or for a specific file. You can display the current build environment from the Fix+Continue pulldown menu, the command line, or the Fix and Continue Status Window. When you finish a Fix and Continue session, you can unset the build environment.

Tracking Program Output

A successful run results in output in the Execution View. This functionality is the same as it is in the Debugger without Fix and Continue.

Chapter 3

Using Fix and Continue: A Sample Session

This chapter provides an interactive sample session that demonstrates most of the Fix and Continue functions. The session outlines common tasks you can perform with Fix and Continue, using example C++ application source to illustrate the use of each function. For complete reference information on the Fix and Continue user interface, see Chapter 4, “Fix and Continue Reference.”

Most steps in the session use the graphical interface but give the command-line alternatives.

This chapter contains the following sections:

- “Setting Up the Sample Session”
- “Redefining a Function”
- “Setting Breakpoints in Redefined Code”
- “Viewing Status”
- “Comparing Original and Redefined Code”
- “Ending the Session”

Setting Up the Sample Session

Figure 3-1 Execution View Icon

For this tutorial, use the demo files in the directory */usr/demos/WorkShop/bounce*, which contain the complete source code for the C++ application *bounce*. To prepare for the session, you first need to create the fileset, then launch Fix and Continue from the Debugger. You must enter the commands listed below:

1. **`cd /usr/demos/WorkShop/bounce`**
2. **`make bounce`**

3. **`cvd bounce &`**

   The *cvd* command brings up the CaseVision Debugger, from which you can use the Fix and Continue utility. The Execution View icon (shown in Figure 3-1) and Main View (shown in Figure 3-2) appear. Note that the Debugger shows that the source code status indicator is (`Read Only`).

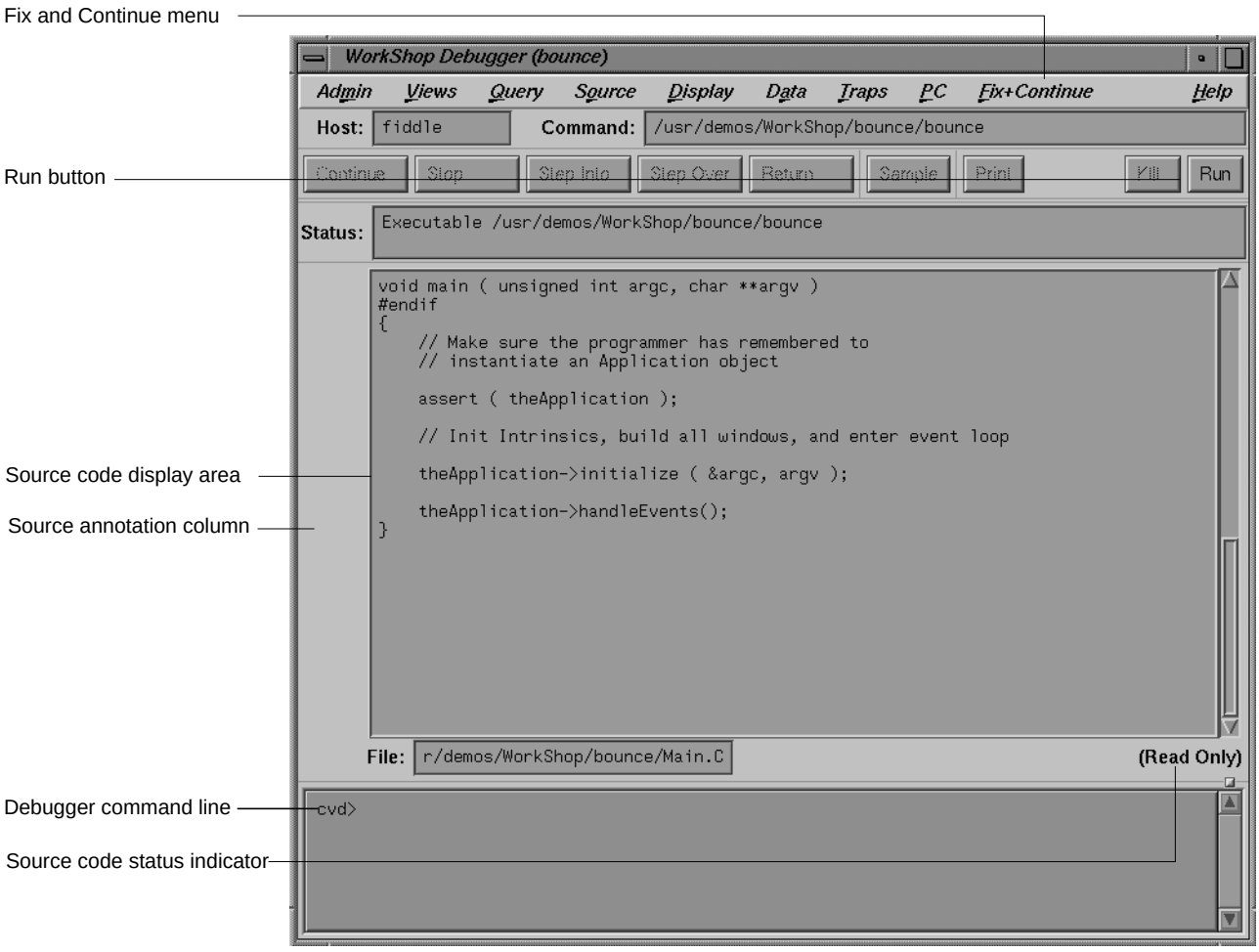


Figure 3-2 Debugger Main View With Fix and Continue Menu

4. Open the Execution View and position the window so you can see it and the Debugger Main View.
5. To see what the program does, click *Run*. The bounce program opens a window on your desktop. Click *Run* in the new window, and then add balls from the Actors Menu to see how the program executes. (You may need to resize the *bounce* window.)
6. The Execution View shows the program output (see Figure 3-3).

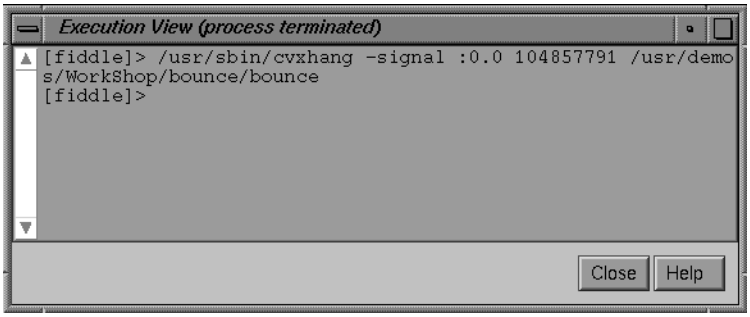


Figure 3-3 Program Results in Execution View

If your screen shows different results, the program files may have been modified during a previous tutorial session.

Redefining a Function

In this section, you will do the following:

- edit a function
- change the code of an existing function and then parse and load the function, rebuilding your program to see the effect of your changes on program output (without recompiling)
- save the changed function to its own separate file

Editing a Function

1. Choose a function to edit by entering the following on the command line:

   ```
   cvd> func Clock::speedChanged
   ```

   This opens the file *Clock.C*, and places the cursor at the beginning of the function **Clock::speedChanged**, as shown in Figure 3-4.

```
 82
 83 void Clock::speedChanged(int value)
 84 {
 85     _delta = 1000 / value;
 86
 87     if ( _id )
 88     {
 89         XtRemoveTimeOut(_id);
 90         _id = XtAppAddTimeOut( XtWidgetToApplicationContext( _w ),
 91                                _delta,
 92                                &Clock::timeoutCallback,
 93                                (XtPointer) this);
 94     }
 95 }
 96
 97 void Clock::speedChangedCallback(Widget, XtPointer clientData, XtPointer call
 98 {
 99     XmScaleCallbackStruct *cb =  (XmScaleCallbackStruct *) callData;
100     Clock * obj = (Clock *) clientData;
101
102     obj->speedChanged(cb->value);
103 }
104
105
```

Figure 3-4 Selecting a Function for Redefinition

2. Show line numbers by selecting “Show Line Numbers” from the Debugger Display menu.
3. Select “Edit” from the Debugger Fix+Continue menu, or enter the Ctrl-E keyboard accelerator. The function is highlighted.
4. Note the results as shown in Figure 3-5. Line numbers change to a decimal notation based on the first line number of the function body. The function body highlights to show that it is being edited. The line numbers of the rest of the file are not affected.

Figure 3-5 Redefined Function

Changing Code

1. To increase the speed of the ball, change the value of *_delta* from `1000 / value` to **`100 / value`**.
2. Click the *Stop* button in the Debugger to halt the *bounce* process.
3. Select “Parse and Load” from the Debugger Fix+Continue menu, or enter the Ctrl-P keyboard accelerator.

   If there are any errors, the Fix+Continue error messages window opens as shown in Figure 3-6. The Debugger command line also gives a report. If all went as planned, there are no errors or warnings.

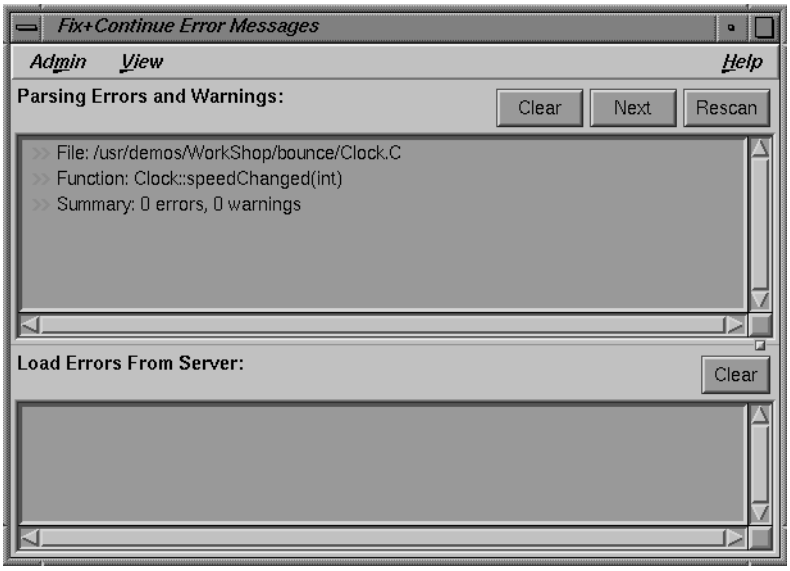


Figure 3-6 Checking Syntax Opens Fix and Continue Status Window

If you do have an error, correct it and repeat steps 1-3. You can go to the error location by double-clicking the appropriate line in the error message window. When you see the change ID and activated status, as shown in Figure 3-7, continue with the next step.

When the parse and load has completed, the highlighting color of the function changes.

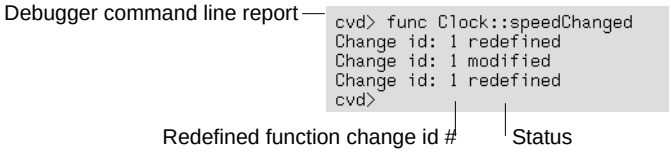


Figure 3-7 Report of Successful Redefinition

4. Select *Continue* from the Debugger main view.
5. The new value is not active until the function is called. To call the function, adjust the slider bar in the *bounce* window (see Figure 3-8).

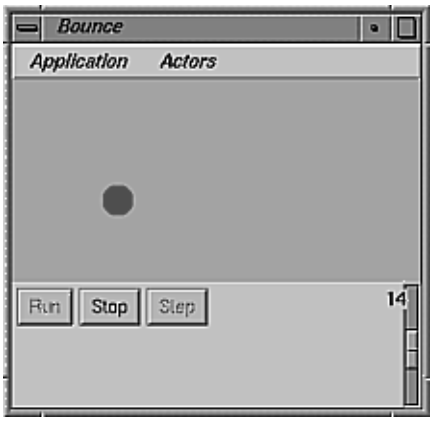


Figure 3-8 bounce Window

Deleting Changed Code

If you make a mistake, there's a graceful way out. Suppose for example that you decided you didn't want to change the speed after all. To delete the change, you need merely select the "Delete All Edits" option from the Fix+Continue menu in the Source view.

Changing Code From the Debugger Command Line

As an alternative to using the Fix and Continue menu, you can redefine and check syntax for a function from the Debugger command line. Try changing *_delta* to 100 by entering the following at the command line:

```
cvd> replace_source "Clock.C":83
"Clock.C":84.0>
"Clock.C":84.1>
"Clock.C":84.2> _delta = 100 / value;
"Clock.C":84.3> .
```

This generates the following output:

```
Change id: 4 redefined
Change id: 4 modified
Process 5779 stopped at ["select.s":12, 0x0fac2010]
Change id: 4 activated
Change id: 4 , build results:
  4  enabled  /usr/demos/WorkShop/bounce/
Clock.C  Clock::speedChanged(int)
cvd>
```

If you prefer to use the command line, experiment with **`add_source`** and **`redefine`** to get the same functionality described for the menu commands. For details on each command, refer to "Command-Line Interface" on page 56.

Saving Changes

Your original source files are not updated until the changed source file is saved. You could save redefined function changes to *Clock.C*. However, if you did, the file would not match the tutorial. So just observe the following steps:

1. Select "Save Files+Fixes As..." from the Fix+Continue menu.
2. Look at the features of the dialog box (see Figure 3-9) that enable you to save your file. To save the changes back to the original source files, click that radio button and then click *Apply* or *OK*. To save to a different file, click the other radio button, choose a suffix, and click *Apply* or *OK*. Since you don't want to save the changes, press *Cancel*.

   Alternatively, on the Debugger command line, you could type **`save_changes -file Clock.C Clock.C`**. Either method saves all the changes to the file, replacing the compiled source code.

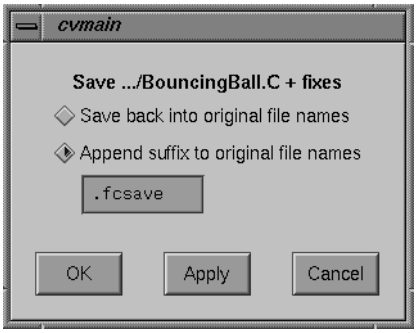


Figure 3-9 Saving a Function File

You usually want to wait until you are finished with Fix and Continue before you save your changes. In addition to the method described above, you can also save your changes with the “Update All Files...” option of the Fix+Continue menu. See “Fix and Continue Menu Operations” on page 36 for more information.

Setting Breakpoints in Redefined Code

To see how the Debugger works with traps (breakpoints) in redefined code you’ll set breakpoints, run the Debugger, and view the results.

1. Choose the function **BouncingBall::BouncingBall** by entering the following on the command line:

   ```
   cvd> func BouncingBall::BouncingBall
   ```

   This opens the file *BouncingBall.C*, and places the cursor at the beginning of the function **BouncingBall::BouncingBall**.

2. Select “Edit” from the Fix+Continue menu or enter Ctrl-E.

3. Enter the following line after line 35:

   ```
   #define SIZE 15
   ```

   This makes the size of the balls smaller.

4. Select “Parse and Load” from the Fix+Continue menu.

5. Set a breakpoint just after your new SIZE definition by clicking in the source annotation column at line 35.3.

   Alternatively, you can set a breakpoint through the command line by entering **`stop at`** # or **`b`** # where # is the line number at which you want your breakpoint. (See Figure 3-10.) Note that in code that has already been parsed and loaded, the line number is in decimal notation.

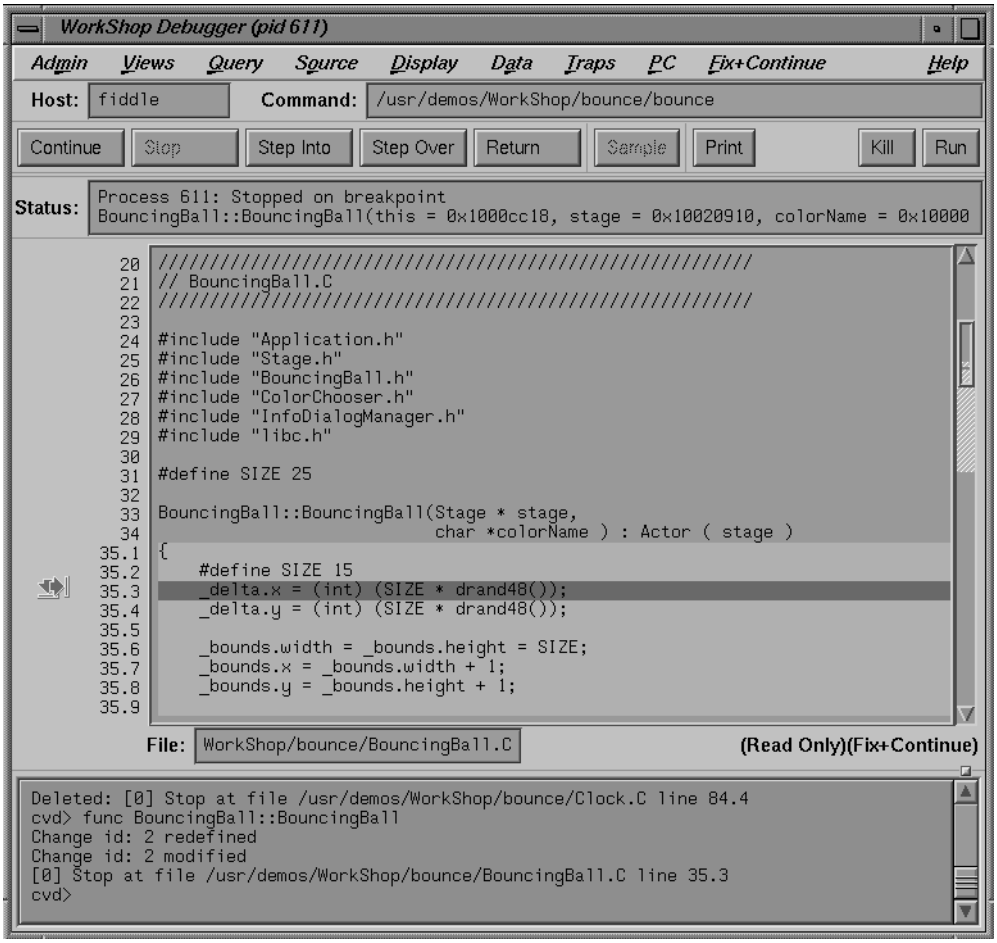


Figure 3-10 Stopping After Breakpoints in Redefined Code

6. Select *Run*, then in the bounce window pull down the Actors menu and select “Add Red Ball”. The Debugger command line reports that the process stopped at some point in the code. You see the following information in the Debugger command line:

```
[1] Stop at file /usr/demos/WorkShop/bounce/
BouncingBall.C line 35.3
[0] Process 595 stopped at [“BouncingBall.C”:35,
0x004088d0]
```

7. Select “Call Stack” from the Views menu to view the results of the breakpoint (see Figure 3-11).

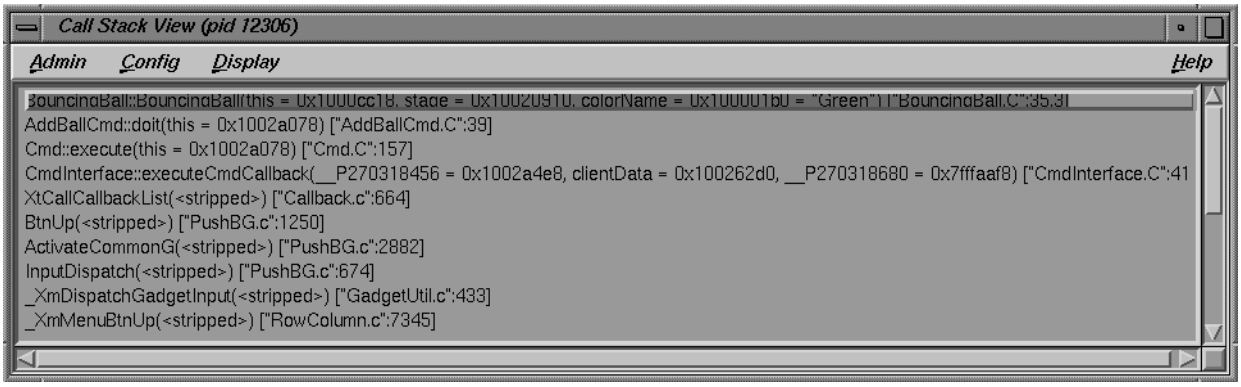


Figure 3-11 Call Stack BreakPoint Results

8. Select “Trap Manager” from the Views menu to view the locations of the traps (see Figure 3-12).

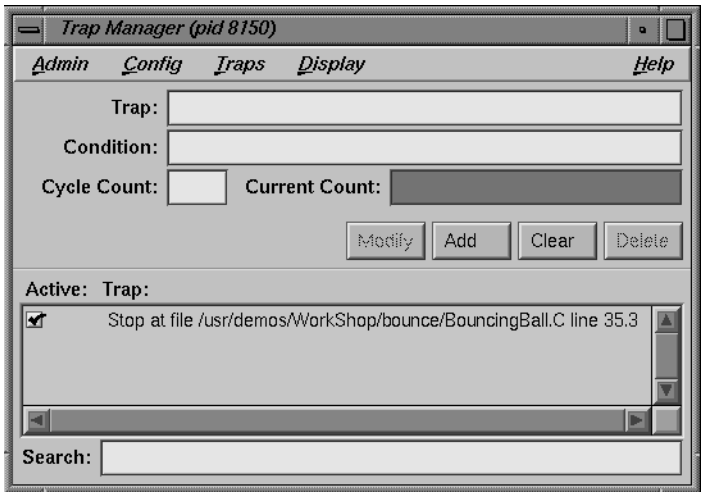


Figure 3-12 Trap Manager BreakPoint Results

9. Remove the breakpoint by clicking on it in the source annotation column.

Viewing Status

Pull down the Fix+Continue menu, choose the Views submenu, and select “Status Window”. The View Status window opens, as shown in Figure 3-13.

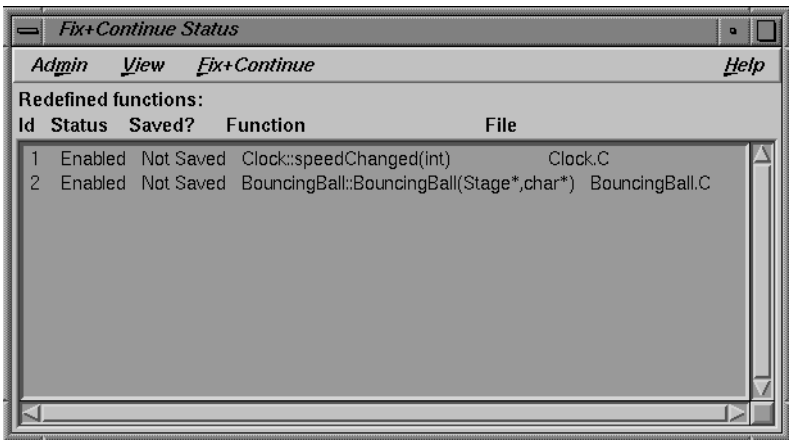


Figure 3-13 Using the View Status Window

Comparing Original and Redefined Code

You can compare your modified code to the original source when using Fix and Continue. This section shows you several ways to view your changes.

Switching Between Compiled and Redefined Code

If you want to see how the redefined code makes your executable different, follow these steps:

1. Select *Run* to view your redefined code. Notice that the balls you add are smaller in your modified version.
2. Place the insertion point in function **BouncingBall**.
3. Select “Edit<-->Compiled” from the Fix+Continue menu. This disables your changes.
4. Select *Continue.* Notice that the balls you add are now their original size, and that the Debugger command line states that the change has been deactivated.

You can get the same results by entering the command **disable_changes** # from the Debugger command line, where # is the redefined function ID number.

To re-enable your changes, do the following:

5. Select *Stop*.
6. Select "Edit<-->Compiled" from the Fix+Continue menu. This re-enables your changes. The balls you add will now be smaller.

   You can get the same results by entering the command **enable_changes** # at the Debugger command line.

Comparing Function Definitions

1. Place the insertion point in the **BouncingBall** function body.
2. Pull down the Fix+Continue menu, choose the Show Difference submenu, and select "For Function". A *xdiff* window opens as shown in Figure 3-14.

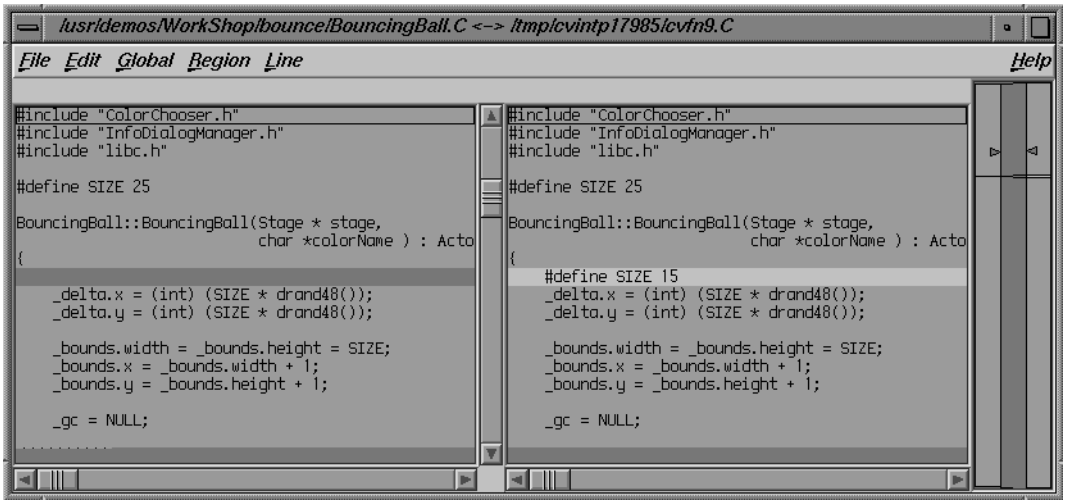


Figure 3-14 Comparing Compiled vs. Redefined Function Code: *xdiff*

You can get the same result by entering the command **`show_diff`** *#* from the Debugger command line.

If you don't like *xdiff*, you can change the comparison tool by pulling down the Fix+Continue menu, choosing the Show Difference submenu, and selecting "Set Diff Tool...".

Comparing Source Code Files

When you have made several redefinitions to a file, sometimes you need a side-by-side comparison of the entire file. To see how your changes to the file look, pull down the Fix+Continue menu, choose the Show Difference submenu, and select "For Function". This opens a *xdiff* window that displays the entire file, rather than just the function.

You'll get the same result from the Debugger command line if you enter the following:

```
show_diff -file BouncingBall.C
```

As an alternative to pulling down menus using the mouse, you can use mnemonics to select the menu item from the keyboard. After closing the difference window, you'll reopen it. With the insertion point anywhere in the file, enter the following:

Alt-f d f

Ending the Session

Exit the Debugger by pulling down the Admin menu and choosing "Exit".

Chapter 4

Fix and Continue Reference

This chapter describes in detail the function of each window, menu, and display in the Fix and Continue utility's graphical user interface (GUI). In addition, the chapter describes the Fix and Continue commands available on the Debugger command line (see "Command-Line Interface" on page 45). Most commands are available from either interface. You can move from one to the other as you prefer. For a task-oriented introduction to commonly-used functions, see Chapter 3, "Using Fix and Continue: A Sample Session."

With Fix and Continue, the Debugger is in the "Edit" state by default. This means that you may select a function for editing, redefine, check syntax, and run the program. You can work with several redefinitions, selectively enabled or disabled.

This chapter contains the following sections:

- "Graphical User Interface"
- "Changes to Debugger Views"
- "Command-Line Interface"

Graphical User Interface

The Fix and Continue GUI affects several WorkShop windows and provides three more. The Debugger and Source View access the Fix and Continue utility from the menu bar. The results of running redefined code are displayed in the Debugger Execution View. Special line numbers (decimal notation) applied to redefined functions appear in several WorkShop views (refer to "Changes to Debugger Views" on page 41). Fix and Continue comes with three windows devoted entirely to Fix and Continue: Status, Message, and Build Environment. This section describes Fix and Continue menu selections and these windows.

The Fix and Continue menu is available from the Debugger Main View menu bar, as shown in Figure 4-1. The menu selections operate on the selected function or on the file shown in the source view. The Fix and Continue menu is also available from Source View and from the Fix and Continue Status window.

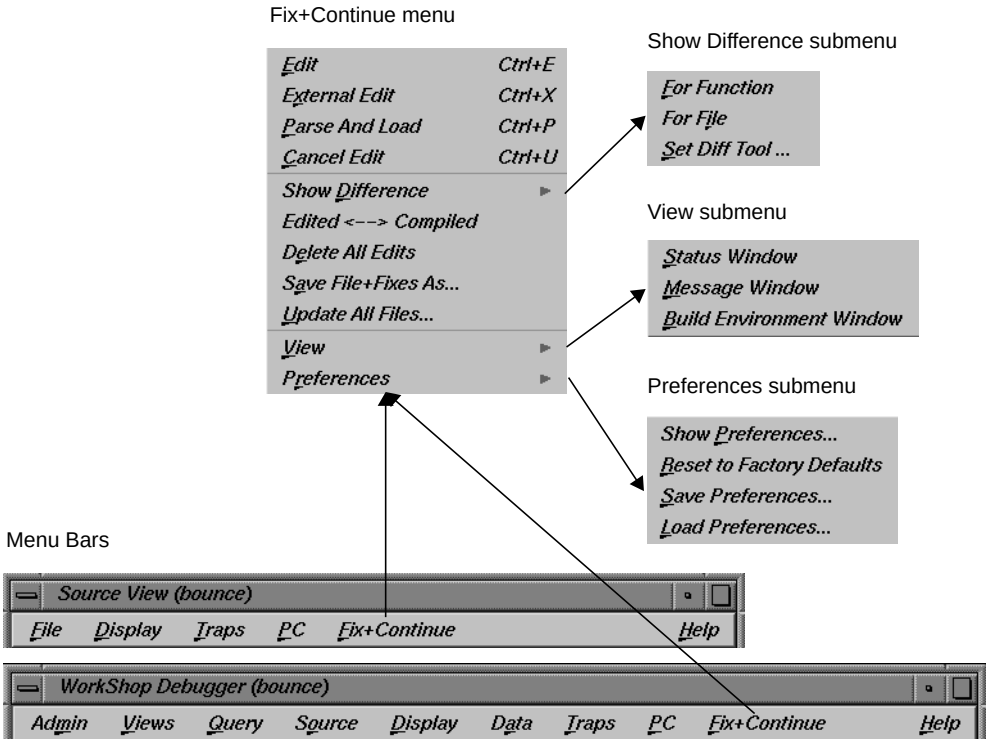


Figure 4-1 Fix+Continue Menu Selections

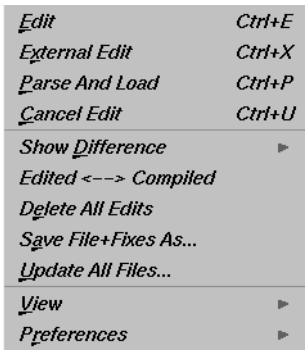


Figure 4-2 Fix+Continue Menu

Figure 4-3 "Save File+Fixes As..." Popup Window

Fix and Continue Menu Operations

The Fix and Continue menu (see Figure 4-2) offers the following menu selections:

"Edit" Allows you to edit functions using the Debugger editor.

"External Edit" Allows you to edit functions using an external editor. The default editor is *vi*, but can be changed by using the "Set Edit Tool..." popup menu in the Admin menu of the Status window. See "Fix+Continue Status Window" on page 34 for further information.

"Parse and Load"
Parses your modified function and loads it for execution. You can execute the modified function by clicking on the *Run* or *Continue* buttons in the Debugger main view.

"Cancel Edit" Takes you out of edit mode.

"Show Difference" submenu
Allows you to see the difference between the original code and your modifications. See "Show Difference Submenu" on page 32 for further information.

"Edited<-->Compiled"
Enables or disables your changes. This switch allows you to see how your application executed before and after the changes you made.

"Delete All Edits"
Deletes any changes that you made to functions.

"Save File+Fixes As..."
Allows you to save your changes to a file (see Figure 4-3). You can save the changes to the current source file (the default), or to a separate file.

"Update All Files..."
Launches the "Save File+Fixes As..." dialog (see Figure 4-3), which allows you to update the current session, saving all the modified functions to the appropriate files.

"View" submenu
: Allows you to change to different views. Fix and Continue supports status, message, and build environment windows. See "View Submenu" on page 32 for further information.

"Preferences" submenu
: Allows you to set your Fix+Continue preferences. See "Preferences Submenu" on page 33 for further information.

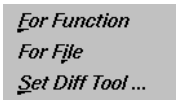


Figure 4-4 Show Difference Submenu

Show Difference Submenu

This submenu(see Figure 4-4) allows you to see the difference between the original and your modified code. It contains the following options:

"For Function"
: Opens a window that shows you the differences between the original function source and your modified source.

"For File"
: Opens a window that shows you the differences between the original source file and your modified version.

"Set Diff Tool ..."
: Launches the Preference dialog (see Figure 4-7), which allows you to set the tool that displays the differences between the two sets of code. The default is *xdiff*. For further information on the Preference dialog, see "Preferences Submenu" on page 33.

View Submenu

This submenu(see Figure 4-5) allows you to open different Fix+Continue view windows. It contains the following options:

Figure 4-5 View Submenu

"Status Window"
: Launches the Fix+Continue Status window. See "Fix+Continue Status Window" on page 34 for more information.

"Message Window"
: Launches the Fix+Continue Message window. See "Fix+Continue Message Window" on page 38 for more information.

"Build Environment Window"
Launches the Fix+Continue Build Environment window. See "Fix+Continue Build Environment Window" on page 39 for more information.

Preferences Submenu

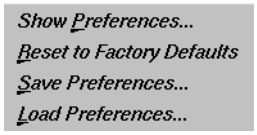


Figure 4-6 Preferences Submenu

The PreferenceMenu (see Figure 4-6) allows you to set various options for Fix and Continue environment, such as the difference tool, the external editor command, and so on. The menu contains the following options:

"Show Preferences"
Launches the Preference dialog (see Figure 4-7), which displays the preferences that are currently enabled for the session, and allows you to change the settings.

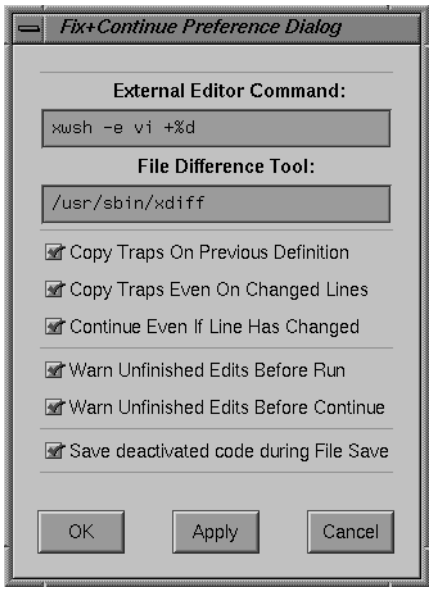


Figure 4-7 Preferences Dialog

"Reset Factory Defaults"
Sets the preferences to the installed defaults.

“Save Preferences”

Allows you to save your preferences to a file. This item brings up the File dialog. See Figure 4-16.

“Load Preferences...”

Allows you to load preferences from a file. This item brings up the File dialog. See Figure 4-16.

Fix+Continue Status Window

This section describes the Fix+Continue Status window (see Figure 4-8). The Status window provides you with a summary of the modifications that you have made during your session. It also allows you quick access to your modified functions, and a somewhat expanded Fix+Continue menu.

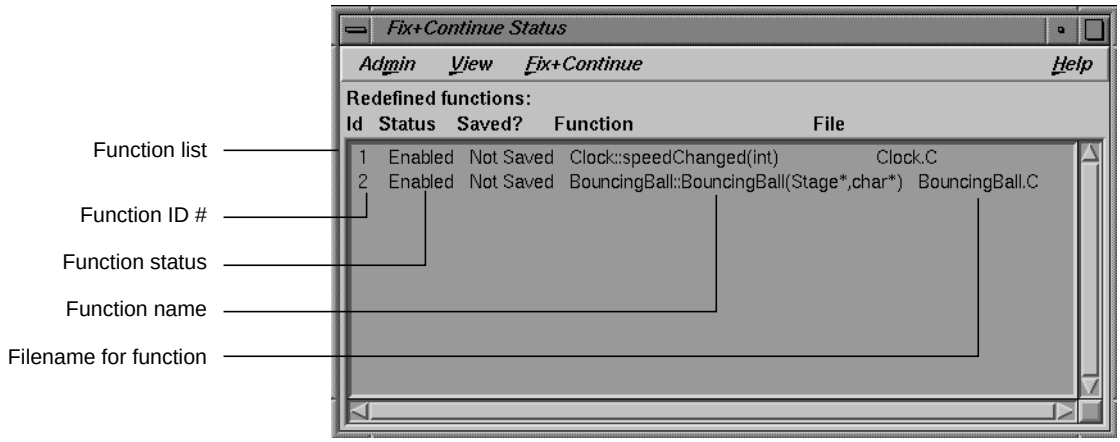


Figure 4-8 Fix+Continue Status Window

The function ID number, status, name, and filename are displayed in the Status window. Double-clicking a line item in the status window brings up the corresponding source in the Debugger main window.

The menus and submenus that provide you with extra functionality through the Status window (see Figure 4-9) are described below.

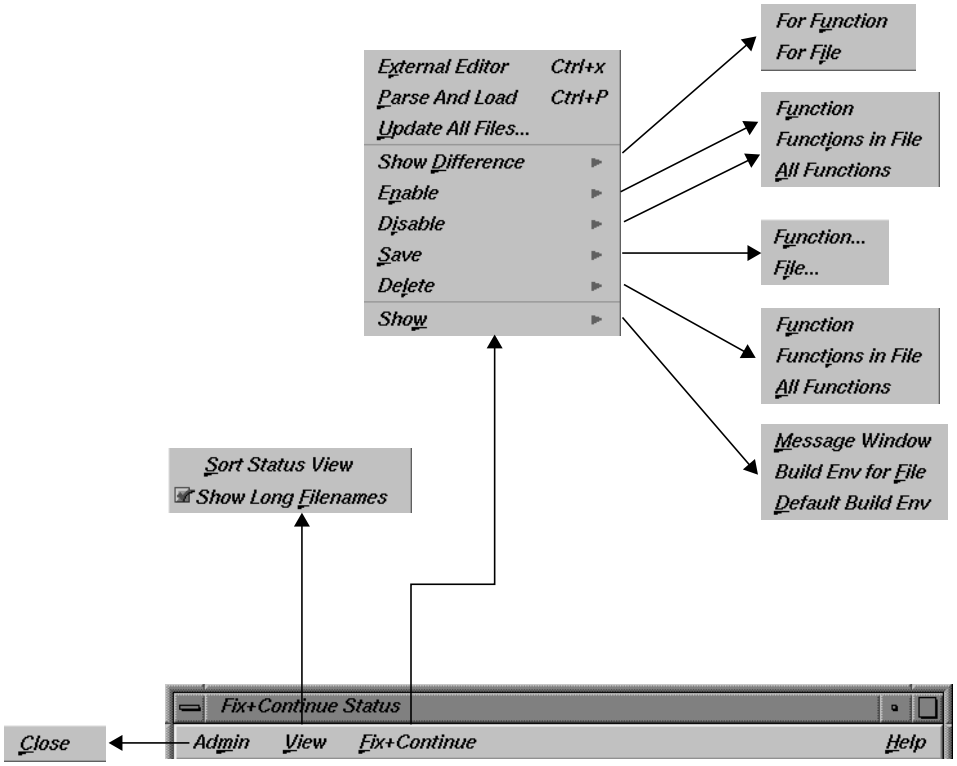


Figure 4-9 Fix+Continue Status Window Menus

Figure 4-10 Status Window Admin Menu

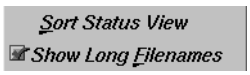


Figure 4-11 Status Window View Menu

Admin Menu

The Admin menu (see Figure 4-10) contains an option for closing the window.

"Close" Closes the Status window.

View Menu

The View menu (see Figure 4-11) contains options for sorting the information in the window, and displaying filenames.

"Sort Status View"
: Sorts the information in the status view according to the field currently selected.

"Show Long Filenames"
: A toggle that allows you to show the absolute (long) pathnames, relative pathnames, or base names.

Figure 4-12 Status Window Fix+Continue Menu

Fix+Continue Menu

The Fix+Continue menu (see Figure 4-12) that is available from the Status view is somewhat different from that available through the Debugger main view. It contains a number of options and submenus, which are all described below. These options and submenus are active on the function that you select in the Source view. You can select a function by clicking on it.

"External Editor"
: Allows you to edit with an external editor such as *vi*, rather than the Debugger's default editor.

"Parse And Load"
: Parses your modified function and loads it for execution. You can execute the modified function by clicking on the *Run* or *Continue* buttons in the Debugger main view.

Update All Files..."
: Launches the "Save File+Fixes As..." dialog (see Figure 4-3), which allows you to update the current session, saving all the modified functions to the appropriate files.

Figure 4-13 Show Difference Submenu

"Show Difference" submenu (see Figure 4-13)
: Allows you to show the difference between the original source and your modified code. You can show the difference in the code in one of the two following ways:

- "For Function" shows the differences for the current function only.
- "For File" shows the differences for the entire file that contains the current function.

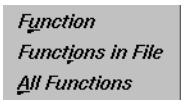


Figure 4-14 Enable Submenu

"Enable" submenu (see Figure 4-14)
: Allowsyou to enable the changes in your modified code in one of the three following ways:

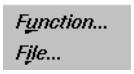


Figure 4-15 Save Submenu

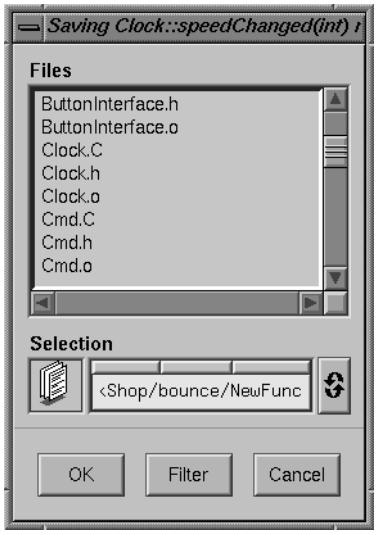


Figure 4-16 File Dialog

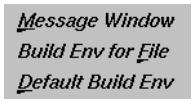


Figure 4-17 Show Submenu

- "Function" enables the changes in the current function.
- "Functions in File" enables the changes to the current function in its own file.
- "All Functions" enables the changes to all functions in the modified code.

"Disable" submenu (see Figure 4-14)

Has the same menu choices as the "Enable" submenu, but disables rather than enables.

"Save" submenu (see Figure 4-15)

Allowsyou to save your code changes to a file. You can save the changes in one of the three following ways:

- "Function..." launches the File dialog (see Figure 4-16), allowing you to save only the current function to a file.
- "File..." launches the "Save File+Fixes As..." popup window (see Figure 4-3), allowing you to save the entire file that contains the current function.

"Delete" submenu (see Figure 4-14)

has the same menu choices as the "Enable" submenu, but deletes rather than enables.

"Show" submenu (see Figure 4-17)

Allowsyou to launch any of the following three different Fix and Continue windows:

- "Message Window" launches a Message window for the selected function. See "Fix+Continue Message Window" on page 38 for more details.
- "Build Env for File" launches a Build Environment window for the file shown in the Source View. See "Fix+Continue Build Environment Window" on page 39 for more details on the Build Environment window.
- "Default Build Env" launches the Build Environment window to show the options that are to be used in cases where they could not be obtained from the target. See "Fix+Continue Build Environment Window" on page 39 for details on the Build Environment window.

Fix+Continue Message Window

The Fix+Continue Message window (see Figure 4-18) contains a list of any errors and other system messages that pertain to your source modifications, parses, and attempts to run your modified source.

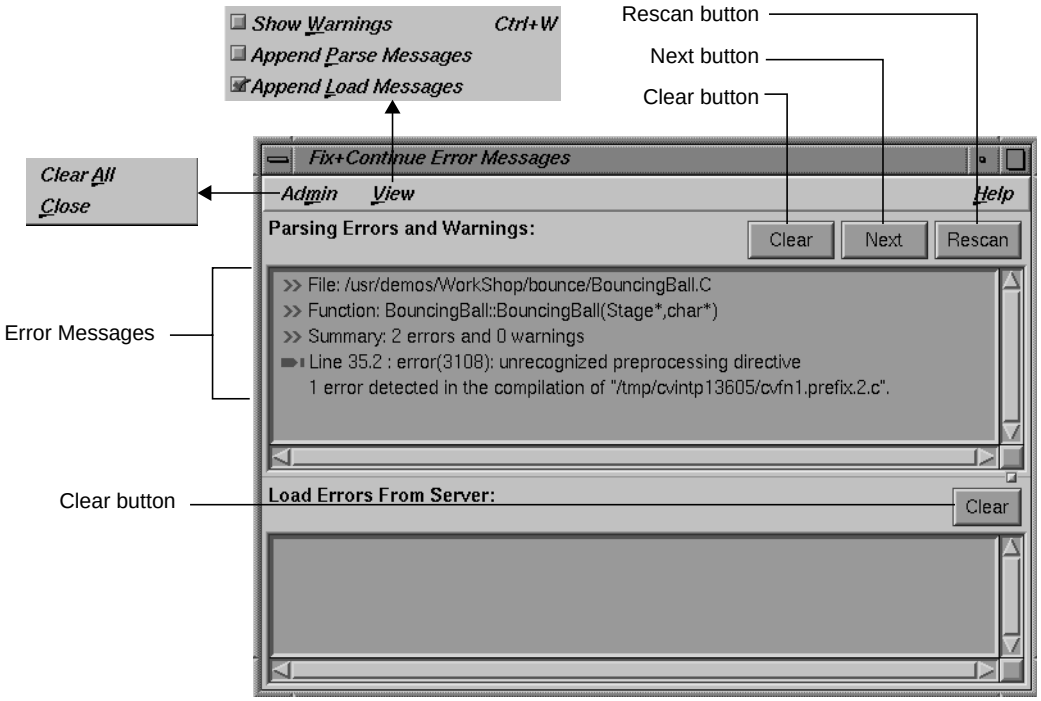


Figure 4-18 Fix+Continue Message Window

You can highlight the source line where the error occurred by double-clicking the appropriate line in the Message window. The window contains the following buttons:

Clear Clears all the parsing errors and warnings.

Next Puts a tick mark on the next unticked error warning entry in the parse messages. It displays the corresponding file and line in the Source view, highlighting it according to the type of error or warning. *Next* doesn't function after all the entries in the messages are ticked.

Rescan Erases all the ticks, so that you can rescan all the error warnings from the beginning.

The added functionality available through the Message window's Admin and View menus is described below.

Admin Menu

The Admin menu allows you to perform either of the following two operations:

"Clear All" Clears all messages in the Message window.

"Close" Closes the window.

View Menu

The View menu allows you to set any of the following three toggles:

"Show Warnings"
Causes compile warnings to be displayed in the parse errors list.

"Append Parse Messages"
Causes parse messages to be appended to the parse errors list.

"Append Load Messages"
Causes load messages to be appended to the load errors list.

Fix+Continue Build Environment Window

This section describes the Fix+Continue Build Environment window (see Figure 4-19). The Build Environment window provides you with the build information for your source code in your current environment. It displays

the command that was used to build your executable and the name of the file that contains the function that you currently have selected.

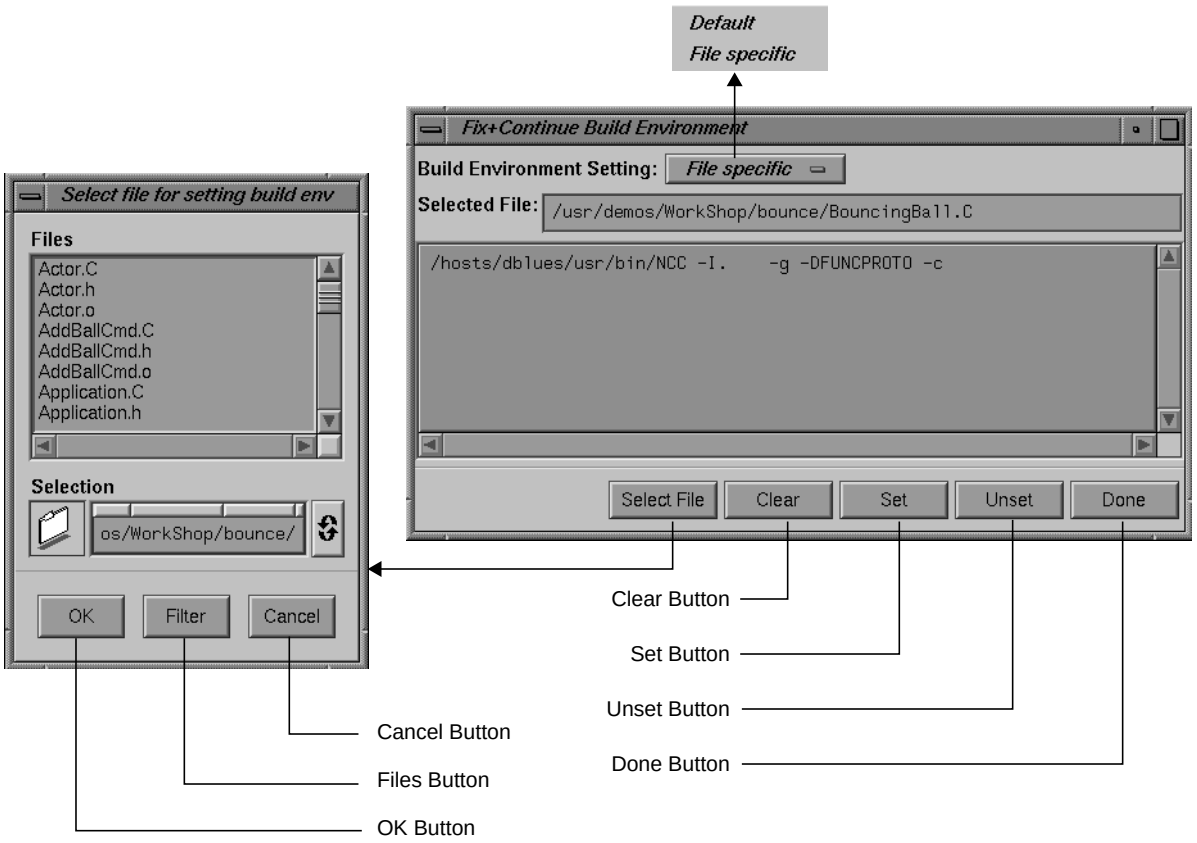


Figure 4-19 Fix+Continue Build Environment Window

The compiler and associated flags that were used to compile the file are normally gathered from the target. You can use the Build Environment window to make any changes to these flags.

The Build Environment window allows you to select your build environment setting through the “Build Environment Setting” toggle, which contains the two options described below:

“Default”	Sets the build environment to default that is displayed in the Build Environment window.
“File Specific”	Sets the build environment to that of the file that contains the currently selected function. You can change the file by clicking the *Select File* button, which launches the File dialog (see Figure 4-16).

The Build Environment window also contains the following buttons:

Select File	Launches the File dialog and allows you to select a file from which to set the build environment.
Clear	Clears the window.
Set	Sets the build environment to what is displayed in the window.
Unset	Unsets the build environment.
Done	Dismisses the window.

Keyboard Accelerators

Use the accelerators in Table 4-1 to issue Fix+Continue commands directly from the keyboard.The accelerators are listed alphabetically by command.

Table 4-1 Fix and Continue Keyboard Accelerators

Command	**Ctrl +** *key*
Cancel Edit	U
Edit	E
External Edit	X
Parse And Load	P

Changes to Debugger Views

When you use Fix and Continue, Debugger views change to show redefined functions or stopped lines containing redefined functions.

Main View

When you open the Debugger after installing Fix and Continue, you'll notice several changes to the environment. All Fix and Continue functions are available through the Fix+Continue menu. See Figure 4-20 for details.

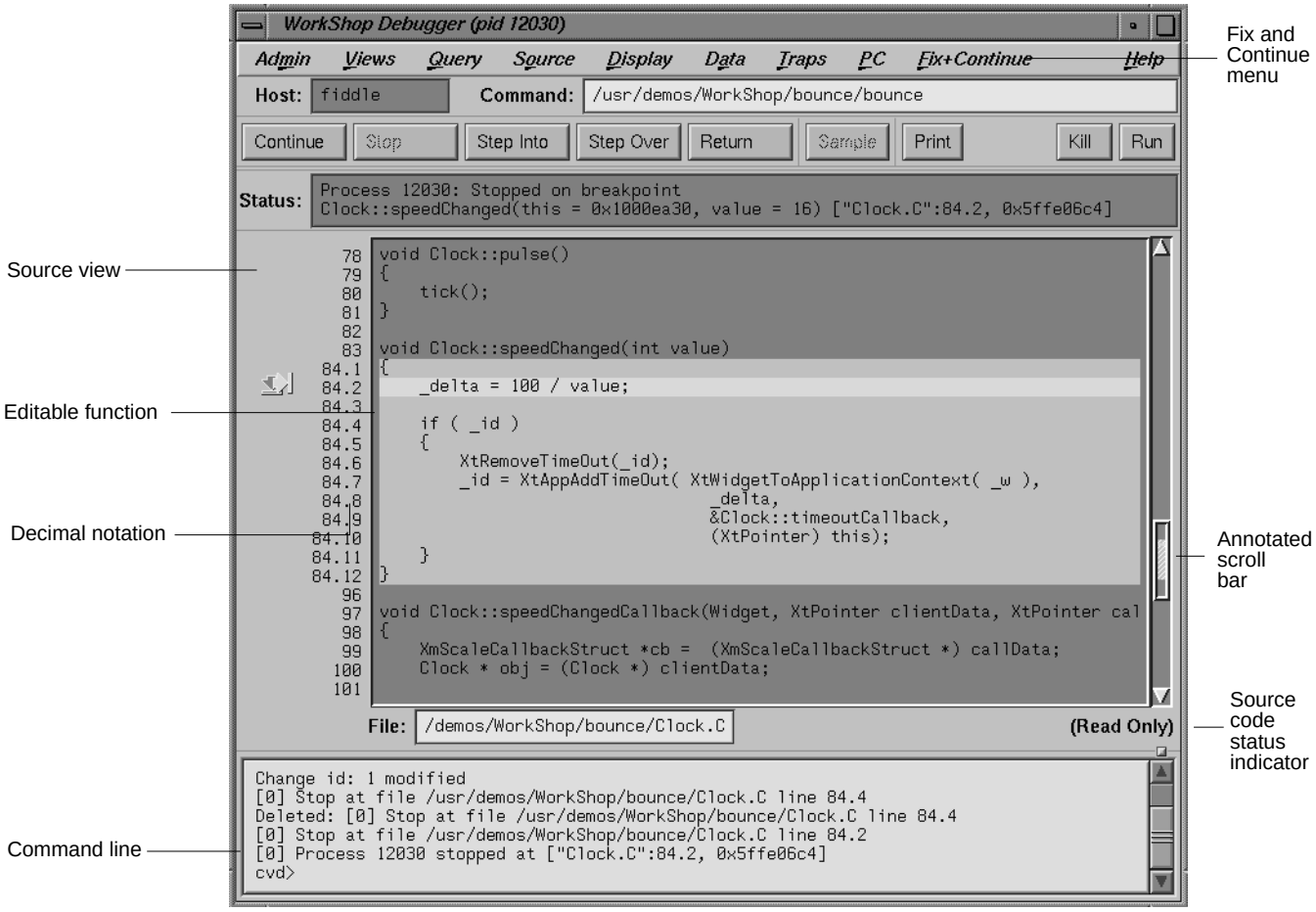


Figure 4-20 Debugger Main View

You select Fix and Continue commands from the Fix+Continue menu or enter them at the Debugger command line. The source code status is Read

Only. Color coding shows the differences between editable code, enabled redefinitions, disabled definitions, and breakpoints. Line numbers in redefined functions have decimal notation that is used for every reference to the line number. The integer portion of the decimal is the same as the first line of the function. This ensures that compiled source code line numbers remain unchanged.

Command-Line Interface

The Debugger command-line interface accepts Fix and Continue commands and reports status involving redefined functions or files. Figure 4-21 shows a function successfully redefined using the command line. Change id 1 was previously redefined and assigned the number 1.

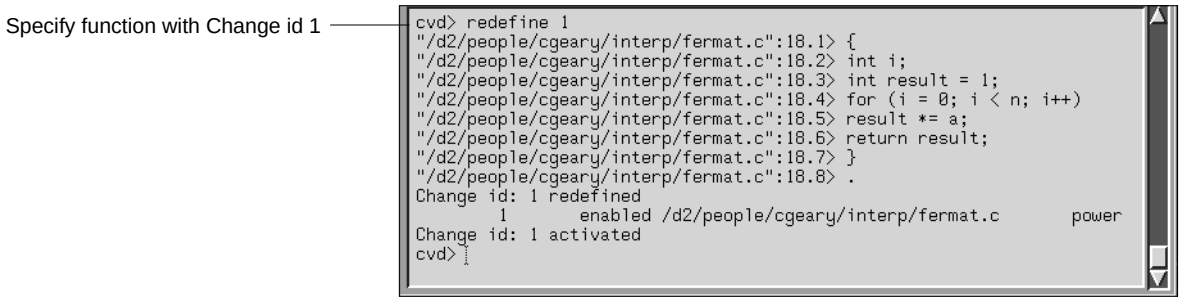


Figure 4-21 Command-Line Interface With Redefined Function

Call Stack

The Call Stack View recognizes redefined functions. It uses the decimal notation for line numbers, as shown in Figure 4-22.

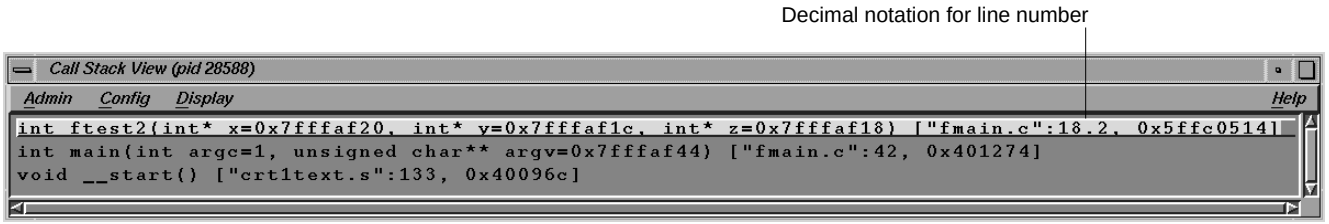


Figure 4-22 Call Stack

Trap Manager

The Trap Manager recognizes redefined functions. It uses the decimal notation for line numbers, as shown in Figure 4-23.

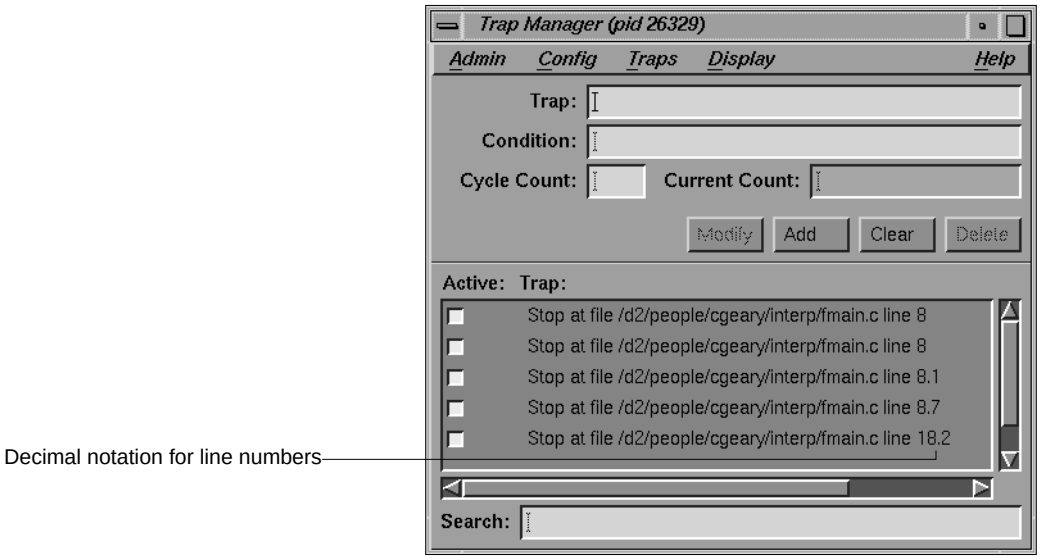


Figure 4-23 Trap Manager With Redefined Function

Command-Line Interface

The commands in this section let you work with Fix and Continue from the Debugger command line. Command arguments that are used for more than one command are grouped and documented separately in the next section. They are listed in alphabetical order for quick lookup.

Common Fix and Continue Command Arguments

This section contains descriptions of some Fix and Continue flags and variables that are common to more than one command.

[-all] Specifies existing changes.

[*change_id*] A unique identifier (ID number) returned on the Debugger command line the first time you redefine a function. From then on, you can use the ID to refer to the function. Disabling or enabling the ID undoes or redoes the cumulative changes performed on the function.

[-file] Specifies the following source file (*filename*); for example: **-file fmain.c**

[*filename*] Specifies a filename with an extension such as *.h*, *.c*, or *.CC*

[*func_spec*] Specifies a function using the following syntax:

[*change_id* | [“*filename*”:] *function_signature*]

where *function_signature* is the name of the function. A C++ member function includes the class name and scope resolution operator (::). For example:

```
3
“fmain.c”: getNumbers
getNumbers
A::bingo
```

[*line_number*]

Specifies the number of the line in the source code.

Fix and Continue Commands

add_source {"*filename*":*line_number*}
Prompts you to add source code lines (for example, **add_source "fmain.c":15.2**). *line_number* must be within the body of a function. Entering a period (.) specifies the end of your input. The source lines you provide are added after the specified line. This command returns an ID existing or new, depending on whether the function affected has already been changed or not. The resulting new definition of the function is executed on its entry next time. See also **delete_source**, **replace_source**.

delete_changes {*func_spec* | -all | {-file *filename*}}
Undoes the changes corresponding to the selected functions (for example, **delete_changes getNumbers -file fmain.c**). Once deleted, you won't be able to use the IDs again, since the IDs associated with the selected functions are released. The default is **-all**. See also **save_changes**.

delete_source {"*filename*":*line_number*[,*line_number*]}
Deletes the given line(s) if ***line_number*** or ***,line_number*** (range) is within the body of a function. An example is: **delete_source "fmain.c":8.6,8.7**. This command eturns an ID existing or new, depending on whether the function affected has already been changed or not. The resulting new definition of the function is executed on its entry next time.

disable_changes {*func_spec* | -all | {-file *filename*}}
Undoes changes specified for the selected functions (for example, **disable_changes getNumbers -file fmain.c**. Nothing happens if the selected function is already disabled. The compiled definition of the function is executed on its next entry. You can invoke this command when the process is stopped or on a running process when a function entry breakpoint is set.

enable_changes {*func_spec* | -all | {-file *filename*}}
Redoes changes specified for the selected functions (for example, **enable_changes getNumbers -file fmain.c**. Nothing happens if the selected function is already enabled. The latest accepted definition of the function is redefined on

its next entry. You can invoke this command when the process is stopped or on a running process when a function entry breakpoint is set.

list_changes [*func_spec* | -all | {-file *filename*}]
Lists one or more lines using the following syntax:

change_id isEnabled filename function_spec

For example:

```
4 enabled foo.c foo
8 disabled A.c++ A::bingo
```

The default is **list_changes -all**.

redefine *func_spec*
[-edit |
{ -read *filename*[*line_number*,*line_number*]}]
Specifies a new body for a function. The new definition is checked, and errors (if any) are printed. The new function body is redefined on the next function entry. Breakpoints (if set) on the old definition are put on the new definition based on their relative line number position from the beginning of the function definition. (Note that some breakpoints may not make it to the new definition.) You can invoke this command when the process is stopped or on a running process when a function entry breakpoint is set. There are three ways to provide a new definition:

- **-edit** pops up an editor of your choice containing the current definition of the function. The specification of the new definition is complete when you exit the editor. You may not leave the editor open. Figure 4-24 shows the *vi* editor.

Figure 4-24 Editing a Function in the vi Editor

- **`-read`** takes the contents of the file specified (within the line numbers if given) as the new function definition.
- No option allows you to type in replacement code from the next line. A period in the first column on a fresh line terminates the definition. For example:

```
redefine getNums
“/usr/fmain.c”:8.1> {
“/usr/fmain.c”:8.2> printf(“In getNums.\n”);
“/usr/fmain.c”:8.3> }
“/usr/fmain.c”:8.4> .
```

  You can use a combination of characters (yet to be determined) to open an editor of your choice containing the lines typed. The specification of the new definition is complete when you exit the editor.

`replace_source {“`*`filename`*`”:`*`line_number`*`[,`*`line_number`*`]}`
Prompts you to type in replacement source if ***`line_number`*** or ***`,line_number`*** (range) is within the body of a function. The source lines you provide replace the specified line(s). An example is **`replace_source “fmain.c”:12`**. This command returns an existing or new id depending on whether the function affected has already been changed or not. The resulting new definition of the function is executed on its entry next time. See also **`add_source`** and **`delete_source`**.

`save_changes {`*`func_spec`*` | {-file `*`filename`*`}}`
`[-[w|a]]`*`filename_to_save`*
Saves (enabled or disabled) function redefinitions or an entire file to a separate file (*`filename_to_save`*). An example of saving a function definition is the following:

```
save_changes getNumbers getNumbersFunc
```

If you specify the **`-file`** option, then before saving to *`filename_to_save`*, all function changes are applied to the compiled source of the file (with the condition that the file has had only its functions redefined, and has not been edited since the last build). An example of saving an entire file is the following:

```
save_changes -file fmain.c fmain.c
```

-w replaces the *filename_to_save*. **-a** appends to the *file_to_save*. An example of adding a function to a file is the following:

```
save_changes getNumbers -a newFuncs
```

See also **delete_changes**.

setbuildenv ["*filename*"] compiler-flag-list

Overrides default build environment flags (compiler options). Without *filename*, the flags are passed along with **-c -g** flags to the compiler for any function in any file except those set separately with **setbuildenv**. An example is the following:

```
setbuildenv -DnameA -Idir
```

If *filename* is given, this command sets separate flags specifically for that file. For example, consider the following:

```
setbuildenv "fermat.c" -DnameB -Ianotherdir
```

See also **unsetbuildenv**.

showbuildenv ["*filename*"]

Lists all the build environment flags set so far. **showbuildenv** "*filename*" lists any build environment specs set separately with **setbuildenv** "*filename*".

show_changes [*func_spec* | -all | {-file *filename*}]

Prints the code of all enabled redefinitions of the specified function(s). The default is **show_changes -all**. See also **enable_changes** and **disable_changes**.

show_diff {*func_spec* | {-file *filename*}}

Launches a *xdiff* comparing the compiled source and its latest redefinition for the specified function. If **-file** *filename* is specified, *xdiff* shows the difference between the compiled file and the file with all redefinitions applied to the compiled source of the file (with the condition that the file has had only its functions redefined, and has not been edited since the last build).

unsetbuildenv ["*filename*"]

Disregards the default build environment flags if specified earlier. For all functions in files that don't have an overriding build environment, **unsetbuildenv** passes only the **-c** and **-g** flags.

If *filename* is given, this command disregards the build environment flags specified for the file earlier. Further redefinition of the functions in the file use the default build environment flags, if set. See also **setbuildenv**.

Chapter 5

Getting Started with the C++ Browser

This chapter is designed to get you up and running with the C++ Browser. It tells you what you need to run the Browser, shows you how to start it, and presents a brief overview of its main windows and menus. It also provides a brief explanation of pertinent concepts concerning C++ class structure. If you are already familiar with C++ and the basic components of the C++ Browser, and want to perform a specific task, see Chapter 6, "Using the C++ Browser: A Sample Session." If you need specific reference information on any part of the Browser's user interface, see Chapter 7, "C++ Browser Reference."

The chapter contains the following sections:

- "Starting the C++ Browser"
- "Understanding C++ Browser Concepts and Components"

Starting the C++ Browser

The C++ Browser is integrated with, and must be launched from, the WorkShop Static Analyzer, *cvstatic*. The Static Analyzer creates a *fileset*, or list of source files, it then uses to build a static analysis *database* used by both the Static Analyzer and the C++ Browser. The Browser accesses this database when it displays C++ class, member, and method information.

Thus, to use the Browser to view your C++ classes, you must perform these tasks:

1. Start the Static Analyzer.
2. Create a fileset that contains the files defining the C++ classes you wish to examine.
3. Build a static analysis database from the fileset.
4. Launch the C++ Browser from the Static Analyzer.

The following sections describe the previous steps in detail.

Starting the Static Analyzer

To start the Static Analyzer, perform these steps in your shell window:

1. Move to the directory containing your source files or your database (unless your fileset has path information to the source directory).

   cd *source_directory*

2. Start the Static Analyzer. If you want to build a database, enter:

   ```
   cvstatic &
   ```

   If you have already built a database, enter:

   ```
   cvstatic -readonly &
   ```

Creating a Fileset

When you start the Static Analyzer, it creates a default fileset that includes all C, C++, and Fortran files in the directory from which you started it. You need to create a custom fileset. The Static Analyzer builds the database from your fileset.

If a fileset exists, select “Change Fileset” from the Admin menu and select files from the Fileset Selection Browser window that appears.

To create a new fileset, perform these steps:

1. From the Static Analyzer’s Admin menu, select “Edit Fileset.”
2. In the Fileset Editor window, first ensure that the Directories field shows the location of your source files. If your source files are in a different directory, click the *Move Directory Parser* button.
3. Click the *C++* toggle under the list of files, to restrict the files shown.
4. Click the *Move Files Parser* button to move the list of files into the Parser Fileset list box. Files in this box are parsed, and the information goes to the static analysis database when you scan.

If you are not satisfied with the files in the *Parser Fileset* list box, remove them all. Select all the lines and then click the *Remove* button. Continue moving directories and files to create the parser fileset.

5. When you are satisfied with the files in the *Parser Fileset* field, click *OK*.

For detailed information on how to create a custom fileset, as well as general information on filesets and their use, see Chapter 4, "Creating a Fileset and Generating a Database," in the *CASEVision/WorkShop User's Guide*.

Creating a Static Analysis Database

If you are recreating the database, rescan using one of the following methods. If you've just modified source code and want to update, select "Rescan" from the Admin menu. To completely rebuild the cross-reference database, select "Force Scan" from the Admin menu.

To build a new database, select "Force Scan" from the Admin menu.

Launching the C++ Browser from the Static Analyzer

To launch the Browser from the Static Analyzer, choose the "C++ Browser" command from the Static Analyzer's Admin menu. The Class View window of the C++ Browser appears, along with the chooser window, List of Classes (see Figure 5-1). The Class View window and other major components of the Browser interface are described in the next section, "Understanding C++ Browser Concepts and Components."

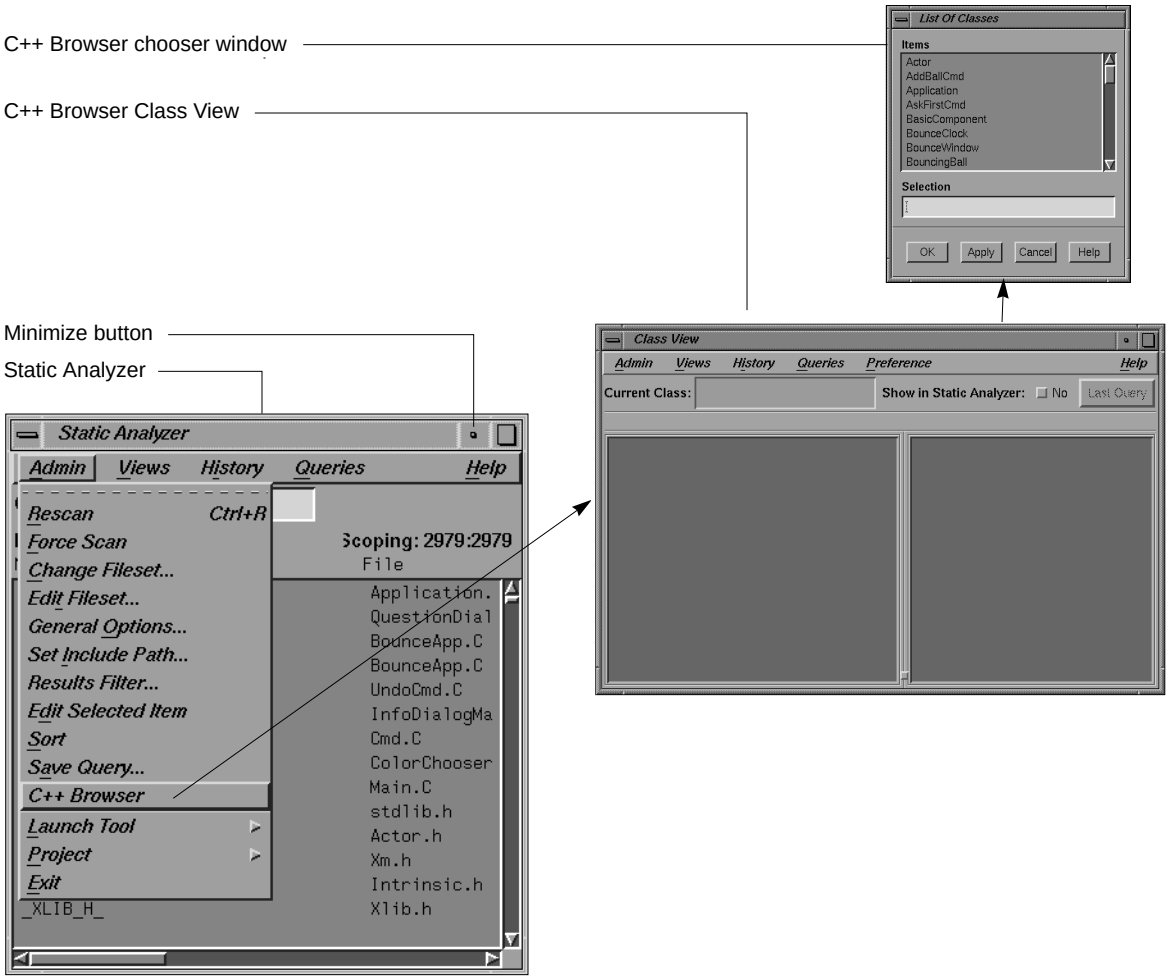


Figure 5-1 Static Analyzer Launches C++ Browser

Understanding C++ Browser Concepts and Components

You can use the Browser to view the C++ classes used in any application or library. Specifically, you view classes present in the sources included in the fileset. The Browser's Class View window (described on page 59) shows the internal structure of a class and its relationships with other classes in a

textual, outline format. The Class Graph window displays the same information as a graphical view of the class structure, providing more of a global perspective of the overall class structure. Thus, you can simultaneously examine the overall class structure, as well as the details of an individual class (its methods, members, friends, and so on).

This section describes the basic user interface design of the C++ Browser. It also introduces the concept of queries, the means by which the Browser and the Static Analyzer let you narrow your search for specific information.

C++ Class Structure and the Current Class

The C++ Browser displays three main types of information:

- the *members* of a class (*current class*) that you select from the set of classes found in the given fileset.
- the classes in the given fileset related to the current class and, conversely, the interclass relationships between the current class and other classes, and global relationships between all the classes found in the given fileset
- the *calling structure* of member functions (methods) belonging to selected classes

This section provides a brief overview and explanation of the different terms the Browser uses to refer to the members, classes, and interclass relationships it displays.

Members

The C++ Browser displays four kinds of class members:

- *types* define data types declared by a class.
- *data* members are variables that contain state information for a class.
- *methods*, or member functions, define how a class interacts with other classes and structures.

- *virtual methods* define how an instantiated object interacts in conditions when parent and child classes have identically named methods that perform different functions. Using virtual methods for an object ensures that the method invoked is defined by the class from which the object was instantiated, regardless of type casting.

Each of these four kinds of members come in two types:

- *static* members, meaning that all objects of a given class contain the same value for a given member, and when that value is changed, it changes for all instances of that class
- *instance* (non-static) members, meaning that the value of the given member can be different for each instance of that class

Finally, each of these kinds of members can fall into one of three categories, based on their accessibility:

- *public* members can be accessed by any method or C-style function
- *protected* members can be accessed only by methods in derived classes, friend classes, or friend functions (see the next section, “Related Classes and Functions”).
- *private* members can be accessed only by methods in the class in which they are defined, friend classes, or friend functions.

The C++ Browser displays class members in a hierarchical, expandable outline format based on the categories described above. The layout of the member list display is customizable (see “Customizing the C++ Browser” on page 128), and the default display is described in “The Class View Window” on page 59 and in detail in Chapter 7, “C++ Browser Reference.”

Related Classes and Functions

The C++ Browser displays this information about class structure from the point of view of a chosen *current class*:

- *base classes*, **meaning the current class and its ancestors**
- *derived classes*, **meaning subclasses of the current class**
- classes whose members the current class *uses* or classes that are components of the current class

- classes that members of the current class are *used by* or classes of which the current class is a component
- *friend classes* and *friend functions* of the current class

Like the member information, the Browser displays related class information in a hierarchical, expandable outline format based on the categories described previously.

The layout of the related class list display is also customizable (see "Customizing the C++ Browser" on page 128), and the default display is described below in "The Class View Window" and in detail in Chapter 7, "C++ Browser Reference."

Interclass Relationships

There are four main types of interclass relationships displayed by the C++ Browser:

- *inheritance*, which describes the relationship of parent classes to derived classes
- *containment*, which describes the relationship of container classes to the classes they contain
- *interaction*, which describes the relationship of classes using methods of other classes.
- *friends*, which describes the relationship of classes declaring other classes as friends.

The Browser displays this information in a graphical tree format as shown in Figure 5-2. For more information on the details of the user interface, see "The Class Graph Window" on page 61 and Chapter 7, "C++ Browser Reference."

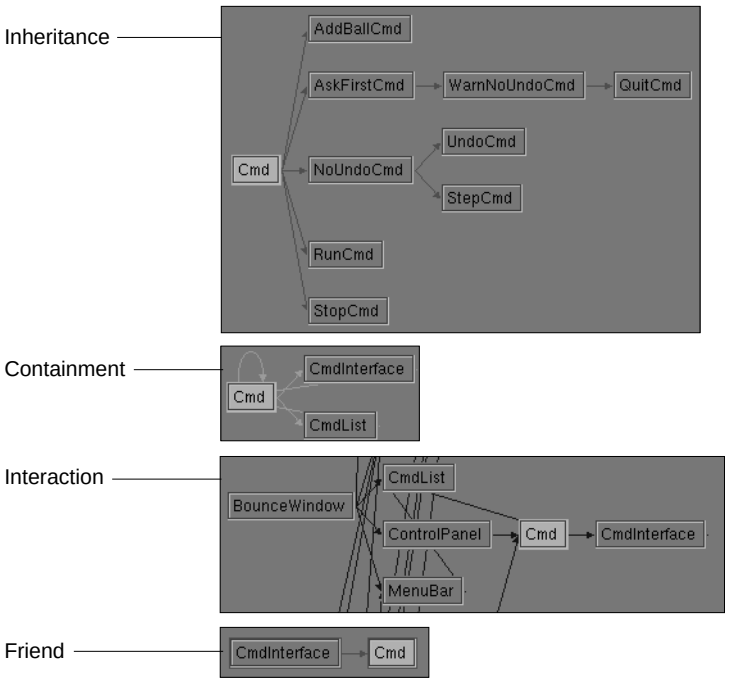


Figure 5-2 Interclass Relationships

Main C++ Browser Windows

The C++ Browser uses three main windows (also called views) to display different sets of information about classes, class members, and interclass relationships.

- The Class View window contains member and class information, organized in an expandable, hierarchical outline format. The Class View window allows you to make queries on listed members and classes, and lets you invoke the other main Browser windows from pop-up menus and the menu bar.
- The Class Graph window contains a graphical display of the current class (as displayed in the Class View window) and related classes found in the static analysis database.

- The *Call Graph* window contains a graphical display of the calling relationships of methods and virtual methods selected from the Class View window.

Some of the features of these windows are described below.

The Class View Window

Class View is the primary C++ Browser window. It opens when you launch the Browser from the Static Analyzer's Admin menu. Figure 5-3 shows an example of the Class View window.

You can use the Class View window to:

- view lists of members defined in the current class and classes associated with the current class in an expandable side-by-side display
- issue queries about a selected member or class to examine details of its structure and how other members or classes relate to it
- open Class Graph and Call Graph windows to further explore class and method relationships
- display detailed results of your queries in the Static Analyzer, or use the annotated scroll bar to find the results of your queries quickly within the Class View member and related class lists
- view source files containing class and member definitions
- generate reference pages

Directly below the menu bar is the Current Class field, an editable text field (with name completion) that contains the name of the current viewpoint class. Below this, the Class View window presents two lists containing information about the currently selected class:

- The *member* list (on the left) gives you a detailed view of the members of the current class.
- The *related class* list (on the right) shows the relative position of the current class in your application's or library's class hierarchy.

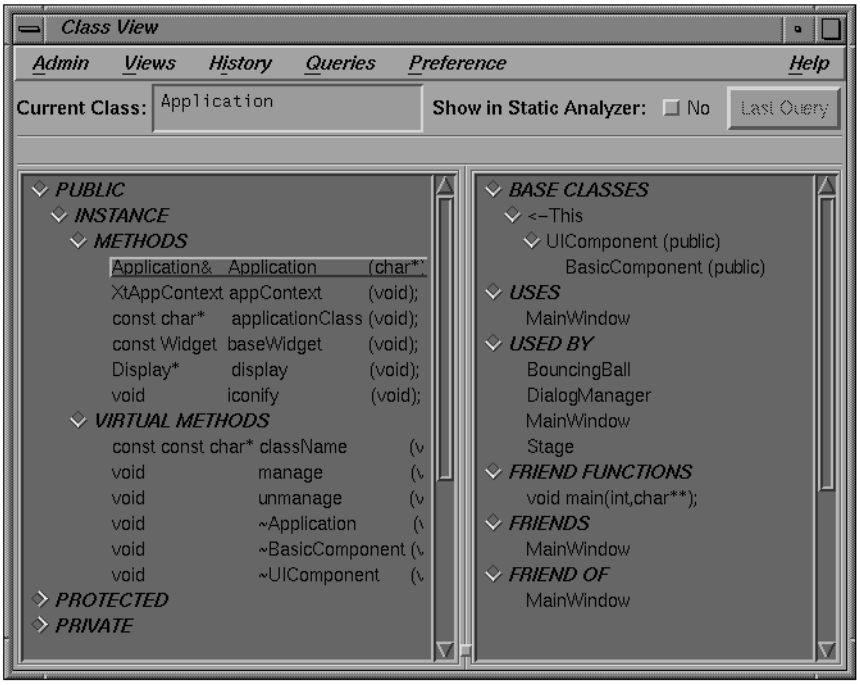


Figure 5-3 The Class View Window

By default, the Browser sorts the members and classes under keyword headings, discussed previously in “C++ Class Structure and the Current Class” on page 55. Ways of altering the sorting scheme are discussed in “Customizing the C++ Browser” on page 128.

The Class View window features annotated scroll bars in its member and related class lists. When you issue a query, the scroll bar contains marks (“annotations”) that indicate the relative location of members or classes identified by the query within each list. For more information on issuing queries, and on using the Class View window in general, see Chapter 6, “Using the C++ Browser: A Sample Session.”

The Class Graph Window

The Class Graph window provides a graphical view of class structure (see Figure 5-4). You can choose to display a directed graph of any of four interclass relationships, and can switch between these graphs at will, or display multiple graph windows, each displaying a different relationship. You can choose to view relationships among all the classes in the static analysis database, or just those that directly involve the class. Double-clicking on a class node in the Class Graph selects the class as the new current class in the Class View window.

Buttons located at the bottom of the Class Graph window allow you to adjust the view of the class structure. You can

- enlarge or shrink the class nodes to fit them into the same-size window
- invoke a graph overview window that serves as a navigational aid by providing a miniature, schematic view of the entire graph you're exploring
- toggle a display of multiple connections between two nodes
- align nodes to clarify the display
- turn the direction of the tree graph by 90 degrees so that the tree graph grows downwards instead of to the right
- choose which one of the four interclass relationships so displayed at any given time

The Class Graph View menu gives you different options for setting the scope of the graph. You can

- show the global relationships of the entire set of classes found in the current fileset
- simplify a complicated graph by showing only related (recursively derived) classes of the current class
- show a butterfly view of the current class, showing only the immediate base and derived classes of the selected class

Figure 5-4 shows an example of the Class Graph window.

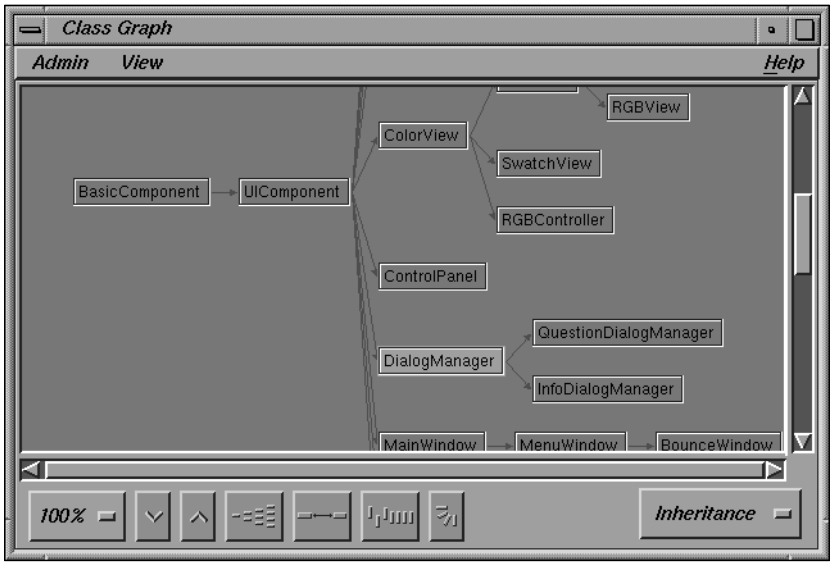


Figure 5-4 The Class Graph Window

The Call Graph Window

The Call Graph window allows you to graphically view the calling structure of class methods and virtual methods. Methods selected from the Class View window can be added or deleted from the Call Graph window. Double-clicking on a method node in the Call Graph window opens a view of the source code defining the method.

Buttons located at the bottom of the Call Graph window adjust the view of the class structure. You can

- enlarge or shrink the class nodes to fit them into the same-size window
- invoke a graph overview window that serves as a navigational aid by providing a miniature, schematic view of the entire graph you're exploring
- toggle a display of multiple connections between two nodes
- align nodes to create a more clear graph

- turn the direction of the tree graph by 90 degrees so that the tree graph grows downwards instead of to the right

Figure 5-5 shows an example of the Call Graph window.

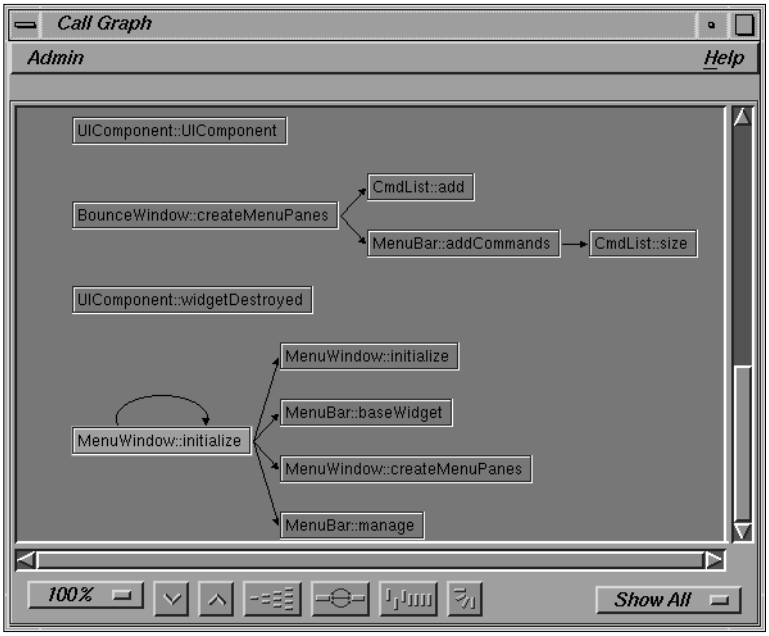


Figure 5-5 The Call Graph Window

Chapter 6

Using the C++ Browser: A Sample Session

This chapter provides an interactive sample session that demonstrates most of the C++ Browser's main functions. The session outlines common tasks you can perform with the browser, using example C++ application source to illustrate the use of each function. For complete reference information on the browser's user interface, see Chapter 7, "C++ Browser Reference."

This chapter contains the following sections:

- "Setting Up the Sample Session"
- "Choosing the Current Class"
- "Using the Class View Outline Lists"
- "Examining Members and Classes"
- "Using the Class Graph Window"
- "Using the Call Graph Window"
- "Generating Reference (Man) Pages"
- "Ending the Session"

Setting Up the Sample Session

The C++ Browser comes with a demonstration directory, */usr/demos/WorkShop/bounce*, which contains the complete source code for the C++ application *bounce*. To prepare for the session, you first need to create the fileset and static analysis database, then launch the browser from the Static Analyzer.

Preparing the Fileset and Database

Prepare for the session by following these steps:

1. Open a shell window and start the WorkShop Static Analyzer in the */usr/demos/WorkShop/bounce* directory:

   ```
   % cd /usr/demos/WorkShop/bounce
   % cvstatic
   ```

   The Static Analyzer window opens.

2. Select “Edit Fileset” from the Admin menu.
3. Click the *C++* toggle under the list of files.
4. Click the *Move Files Parser* button to move the list of files into the Parser Fileset list box.
5. Click *OK*.
6. Select “Force Scan” from the Admin menu.

Launching the C++ Browser

Once the static analysis database is built, you can continue with the steps or close the Static Analyzer. If you have closed it, restart it as follows:

```
cvstatic -readonly
```

The **`-readonly`** option suppresses the Static Analyzer’s Build Window.

Figure 6-1 Minimizing the Static Analyzer

1. Select the “C++ Browser...” command from the Admin menu of the Static Analyzer window (see Figure 5-1). The Class View window opens, along with a chooser window, List of Classes, that contains the names of each C++ class included in the demo directory.
2. Stow the Static Analyzer Window as an icon using the minimize button in the upper right of the window frame; you will be using it later (see Figure 6-1.
3. Position the Class View window in a convenient place on the screen.

You are now ready to begin the sample session.

Choosing the Current Class

The List of Classes chooser window that opened with the Class View contains the complete list of C++ classes included in the current fileset. Select Actor, the first class listed in the List of Classes window, by clicking on it with the left mouse button. Then click on the *OK* button. The List of Classes window closes, and Actor is now listed in the *Current Class:* text field. Information about the class appears in the two outline list views below it (see Figure 6-2). Actor is now the *current class.*

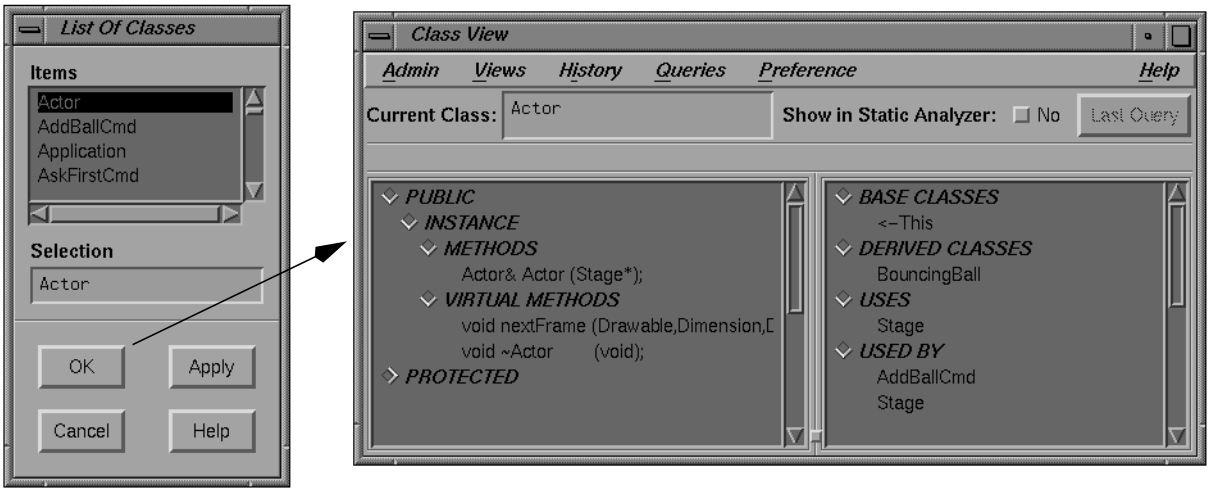


Figure 6-2 Selecting a Current Class

Using Name Completion

Now select the name in the *Current Class:* field by rapidly double-clicking on it using the left mouse button. You want to replace the class Actor with the class MainWindow as the current class. Type

```
Main
```

into the *Current Class:* field, and then press the <space bar>. The *Current Class:* field completes the name of the class for you. Press the **<Enter>** key to set MainWindow as the current class. See Figure 6-3.

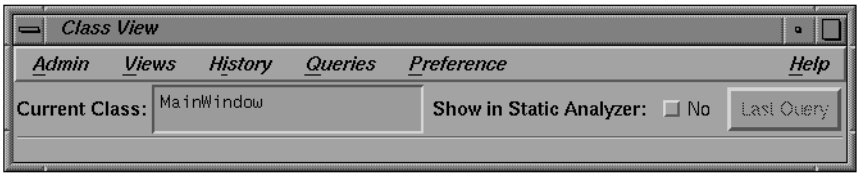


Figure 6-3 Changing the Current Class to MainWindow

Bringing Up the List of Classes Chooser

Without selecting any text, type a question mark (**?**) into the *Current Class:* field. This reopens the List of Classes chooser window. Right now, you want to keep MainWindow as the current class, so click on the *Cancel* button with the left mouse.

Using the Class View Outline Lists

The lower two-thirds of the Class View window contains two side-by-side lists which contain information about the currently selected class in outline form:

- The lefthand, or members, list provides a detailed view of the members of the current class.
- The righthand, or related class list displays the relationships of other classes and of friend functions in your application's or library's class hierarchy to the current class.

Figure 6-4 Expanding a List

By clicking on the outline icon to the left of a heading, you can collapse or expand the list under that category. The direction in which the outline icon points indicates if the heading is expanded or collapsed.

A downward-pointing outline icon indicates that the list is expanded, as shown in Figure 6-4.

Figure 6-5 Collapsing a List

In the member list, click on the ***PUBLIC*** heading's downward-pointing outline icon using the left mouse button. Class View hides all of the entries for that heading, and the icon toggles to become a right-pointing arrow (see Figure 6-5).

Click the ***PUBLIC*** heading's right-pointing icon again. Class View once again displays the entries for that heading, and the icon toggles to become a downward-pointing arrow.

Examining Members and Classes

To learn details about the structure of your C++ code, you make *queries*—predefined questions—about the current class's members and related classes. By making queries, you can gain a detailed view of a large, complicated class structure from the "viewpoint" of any class or member.

Queries search the static analysis database for specific information about classes and their members, including class hierarchy, class and member declarations and definitions, and the interactions among members and classes (for example, which members call which members, where a definition overrides another, where an instance is created or destroyed, and so on).

The C++ Browser answers queries by highlighting items in the member and related class lists that match the query. Optionally, you can display more detailed query results in the Static Analyzer window from which you launched the browser.

In the following sections, you will explore the queries available for data members, methods, and classes, plus several other display features of the Class View window.

Making Queries on Data Members

In this section, you will examine a data member in the class MainWindow (see Figure 6-6).

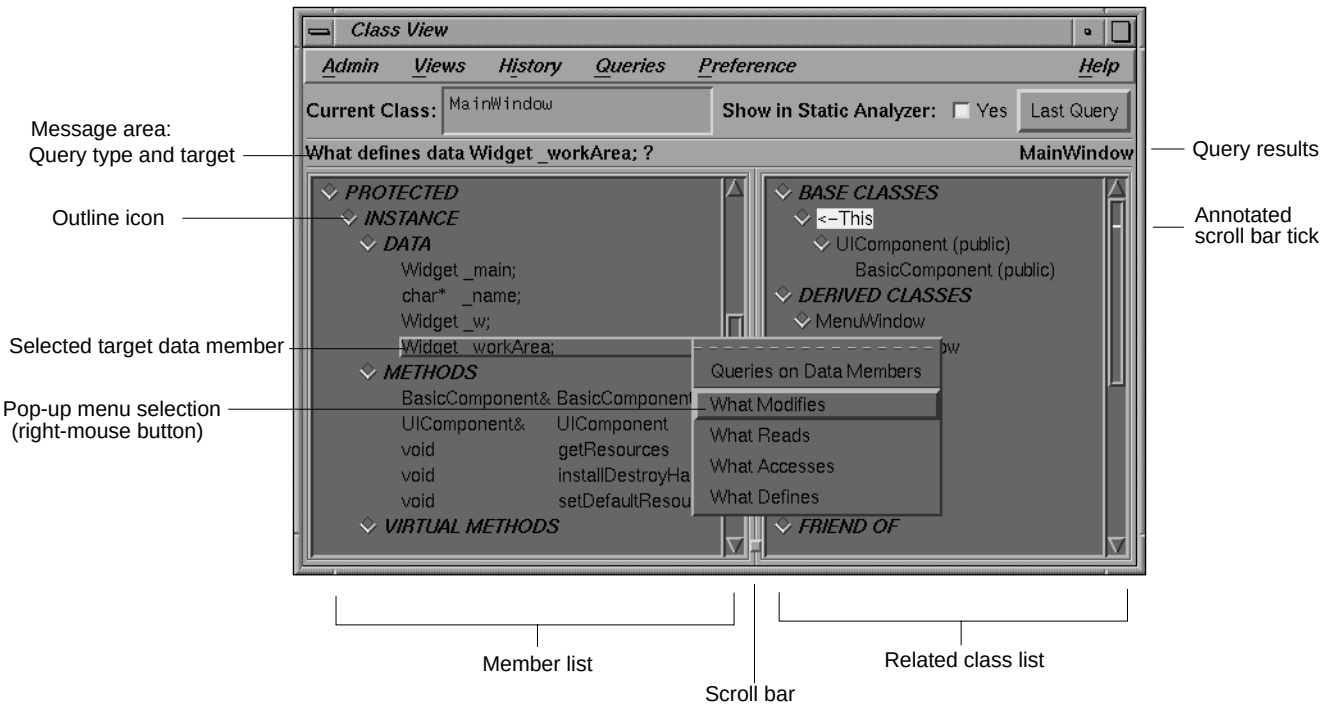


Figure 6-6 Queries on Data Members

To select the data member in the member list, do the following:

1. Use the member list's scroll bar to scroll until the outline heading ***PROTECTED*** is visible (if it is not already).
2. Left-click the corresponding outline icon, and then scroll until the heading ***DATA*** and the members under it are visible.
3. Click on the data member

   `Widget _workArea;`

   using the left mouse button. The member is highlighted with a raised bar.

4. Hold down the right mouse button anywhere in the member list. The Queries on Data Members pop-up menu appears, as shown in Figure 6-7.

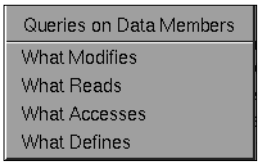


Figure 6-7 Queries on Data Members Pop-Up Menu

Try each of these items and note the results. Methods in the member list and classes in the related class list are highlighted, and the list is automatically scrolled to display the highlighted items.

Also notice the use of the *annotated scroll bars*: they show where highlighted entries occur in the member and related class lists by displaying ticks in the highlight color at the proper location in the scroll bar. When the bottom or top of the scroll bar overlaps a given tick, the corresponding entry is visible in the list window.

The query type, target, and result are shown in the message area between the current class field and the outline lists.

Next, try double-clicking on the member you've selected (*Widget _workArea*). A Source View window opens, highlighting the location of the source for that member (see Figure 6-8). This works for all members in the member list.

Data Member definition

Browser source icon

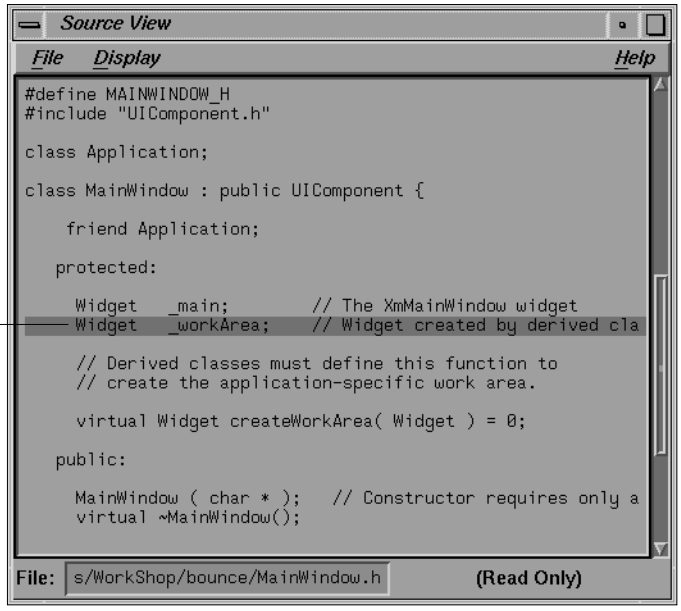


Figure 6-8 Source View Window

Detailed Data Member Query Information

Now, suppose you want to see more detailed information on your queries. Choose “What Accesses” from the Queries on Data Members pop-up menu (see Figure 6-9). Note that the display scrolls to show the results of your query. Then click on the *Last Query* button in the upper right corner of the Class View window.

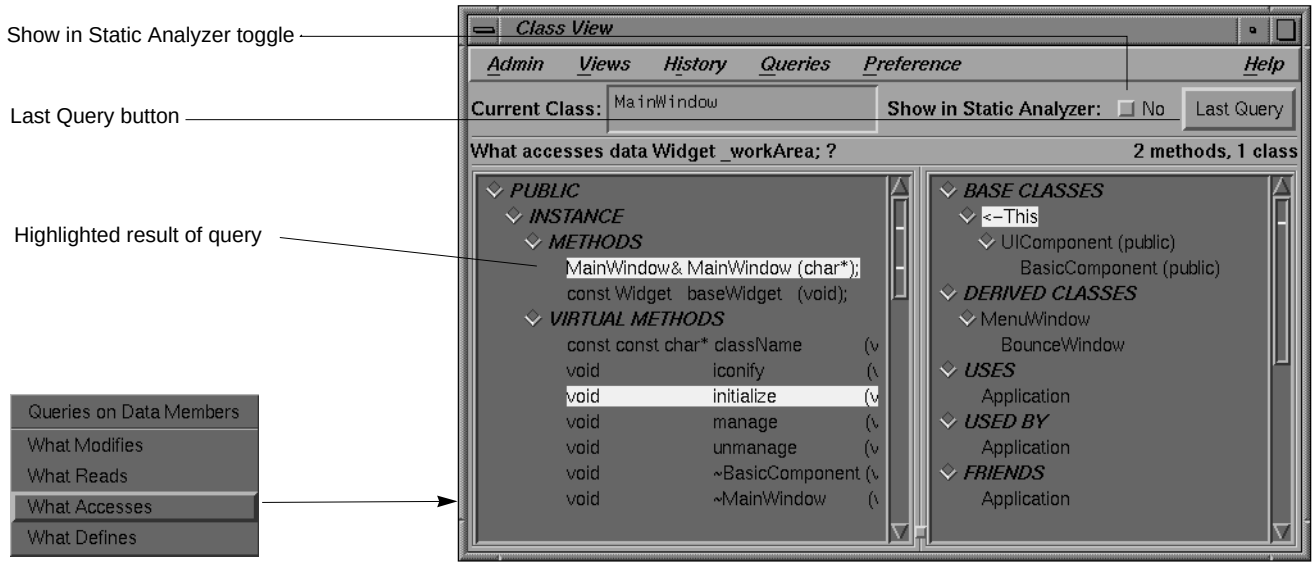


Figure 6-9 "What Accesses" Query Result

Now open the Static Analyzer window that you had minimized earlier. The window contains precise information on where the member is accessed, as shown in Figure 6-10. Using the left mouse button, double-click anywhere in the first entry listed in the Static Analyzer window. A Source View window opens, displaying the highlighted access. Double-click anywhere in any other entry in the Static Analyzer window, and the Source View shifts to the corresponding access.

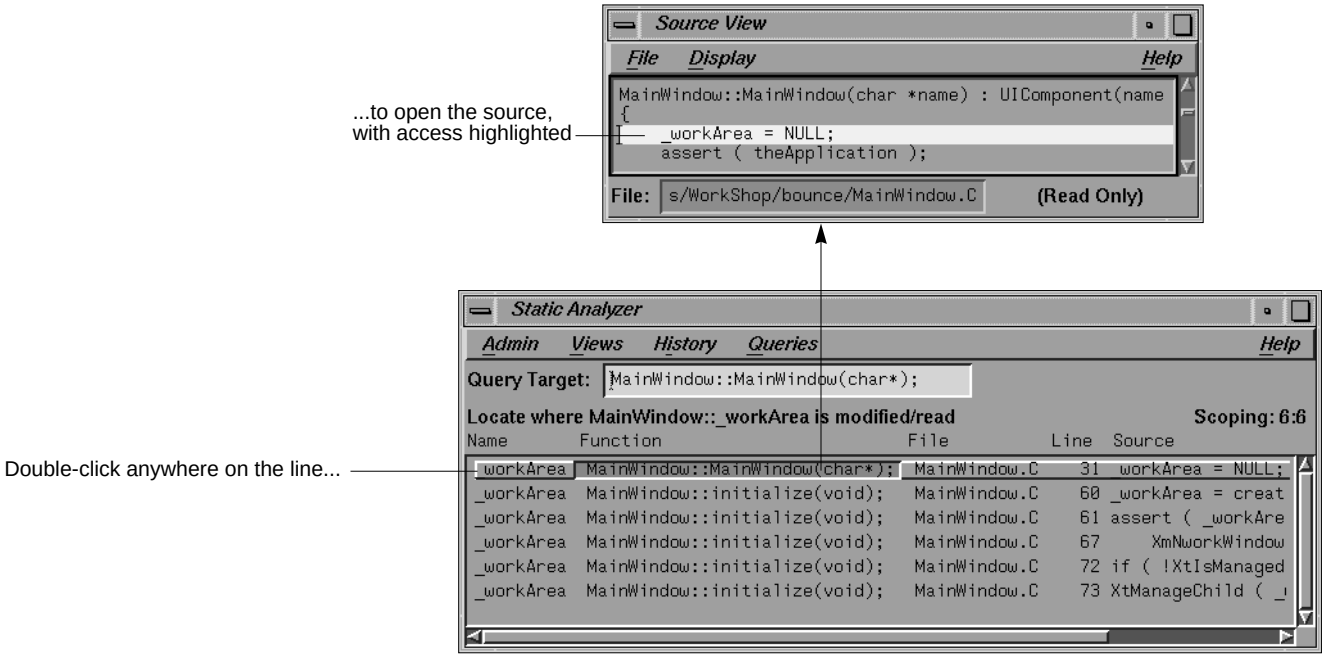


Figure 6-10 Query Result in Static Analyzer and Source View

Try this for each of the other query items, and examine the results.

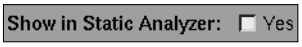


Figure 6-11 *Show in Static Analyzer* Toggle On

When you have finished, return to the browser Class View. Click on the *Show in Static Analyzer* toggle button (see Figure 6-11). The label to the right of the toggle should change to *Yes*. All subsequent queries you make display detailed results in the Static Analyzer window automatically.

Making Queries on Methods

In this section, you will examine a method (member function) in the class MainWindow. To select the method in the member list, follow these steps:

1. Use the member list's scrollbar to scroll until the outline heading ***PUBLIC : INSTANCE : VIRTUAL_METHODS*** is visible.

2. Using the left mouse button, click on the method

   `void initialize(void);`

   The method is highlighted with a raised bar.

3. Hold down the right mouse button anywhere in the member list to bring up the Queries on Methods pop-up menu (shown in Figure 6-12).

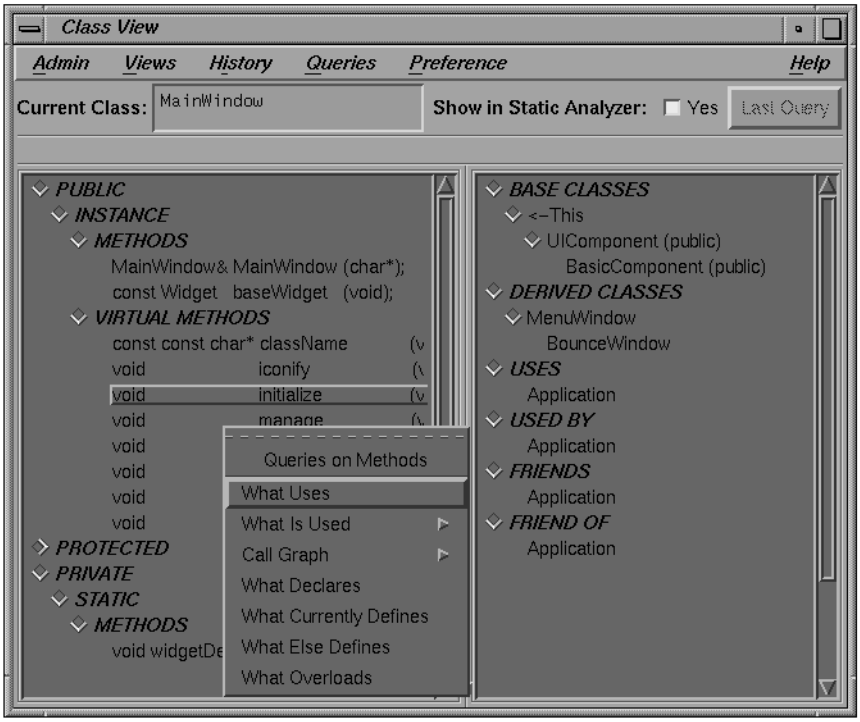


Figure 6-12 Queries on Methods

Try each of the items on the menu (except “Call Graph”), and note the results. Make sure to look at the detailed results that appear in the Static Analyzer window. Also try double-clicking on the method to get a view of its source.

When you are finished, go back and choose “All (method and data access)” from the What is Used submenu (see Figure 6-13).

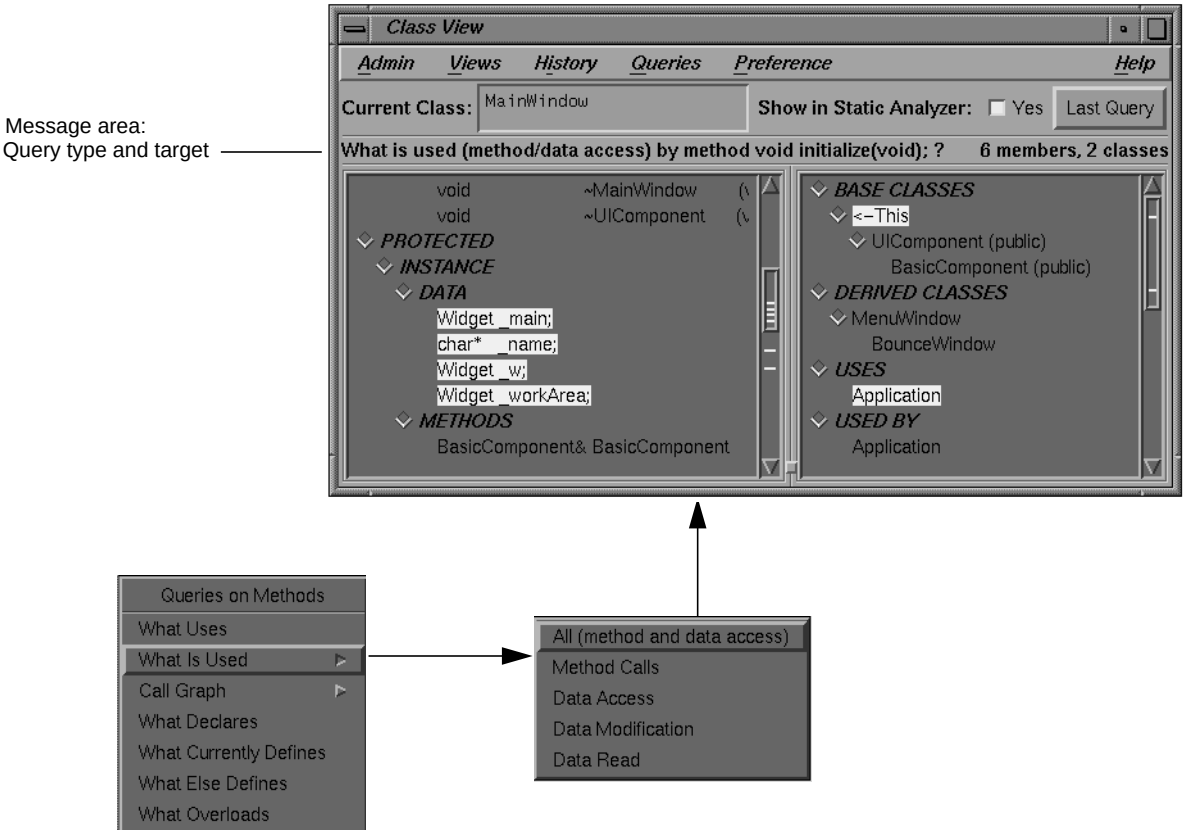


Figure 6-13 "What is Used" Submenu and Query Results

Notice that several items in the heading ***DATA*** (under ***PROTECTED***) are highlighted. Try clicking on the outline icon next to the heading. You'll see that when the list collapses, the outline icon is highlighted. This indicates that there are highlighted items within the collapsed heading. Note that the annotated scroll bar adjusts accordingly.

With MainWindow& MainWindow (char*); selected, try "What Uses" from the Queries on Methods pop-up menu. The Static Analyzer becomes active and displays an error message, as shown in Figure 6-14.

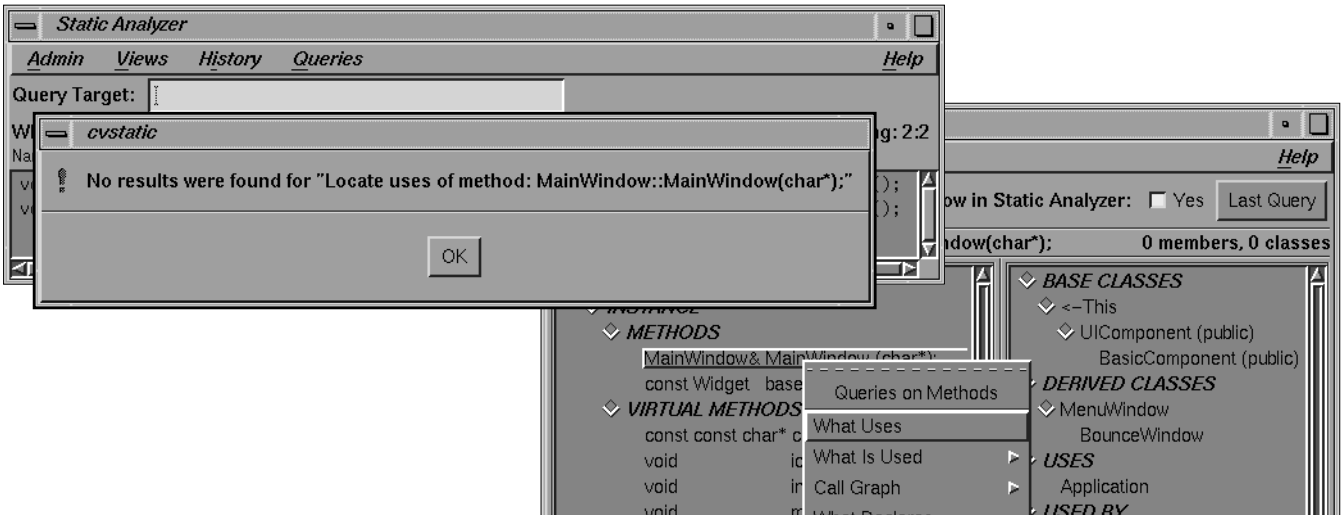


Figure 6-14 No Results Were Found

Making Queries on Parent Classes

In this section, you'll examine the current class (MainWindow) and one of its ancestors. To get ready, scroll the related class list so that the heading ***BASE CLASSES*** is visible. You'll notice that beneath the heading there are class names, each of which is indented from the previous one (see Figure 6-15). The first of these is

```
<-This
```

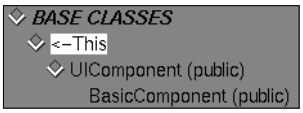


Figure 6-15 BASE CLASSES Representation

which represents the current class. Each class listed below it is an ancestor of the current class; parentage is denoted by indentation. Thus, in this case, UIComponent is the parent of MainWindow, and BasicComponent is the parent of UIComponent. Notice that child classes have outline icons associated with them, allowing you to collapse and expand the hierarchy. Try clicking on the current class outline icon to collapse the hierarchy, and then click again to expand it.

Making Queries on the Current Class

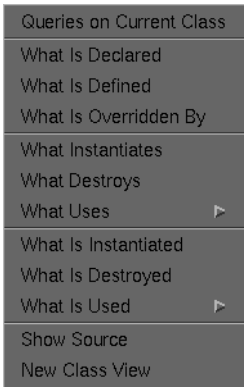


Figure 6-16 Queries on Current Class

Select the current class (<-This) by clicking on it with the left mouse button, then hold down the right mouse button anywhere in the related class list to bring up the Queries on Current Class pop-up menu. The menu is divided into four sections, as shown in Figure 6-16.

Try each of the first three sections of the menu, and examine the results in both the Class View and the Static Analyzer windows. All the queries on the current class in the first three sections are also available from the Queries menu in the Class View menu bar.

The last section of the Queries on Current Class pop-up menu contains two items:

- "Show Source" opens a Source View window on the source or header file containing the declaration of the selected class. The declaration is highlighted in the source. Note that if you minimized a Source View previously, this command updates the window. However, the window remains minimized.
- "New Class View" opens a new Class View window with the selected class as the current class within it.

These two items are found on almost every pop-up Query menu in the related class list, so they won't be mentioned again. Feel free to try them out while you are running this session, but close each new window after trying it so you don't get lost.

Making Queries on Base Classes

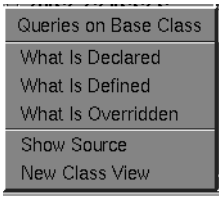


Figure 6-17 Queries on Base Class

Now select the class UIComponent by clicking on it with the left mouse button. Then hold down the right mouse button anywhere in the related class list to bring up the Queries on Base Class pop-up menu (see Figure 6-17).

Try each of the menu items, and examine the results in both the Class View and the Static Analyzer windows.

Making Queries on Derived Classes

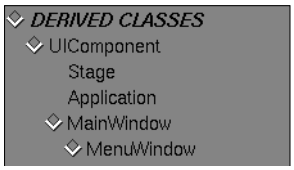


Figure 6-18 DERIVED CLASSES Structure

In this section, you'll examine a class derived from (that is, a descendant of) the current class (MainWindow, see Figure 6-18). To get ready, scroll the related class list so that the heading ***DERIVED CLASSES*** is visible. Notice that beneath the heading there are class names, each of which is indented from the previous one. Each class listed below it is a *descendant* of the current class; childhood is denoted by indentation. Thus, in this case, MenuWindow is the child of MainWindow (the current class is not listed under the heading, however); and BounceWindow is the child of MenuWindow. Note that the meaning of indentation is the opposite of what it was under the ***BASE CLASSES*** heading.

Notice that parent classes have outline icons associated with them, allowing you to collapse and expand the hierarchy. Try clicking on the MenuWindow outline icon to collapse the hierarchy, and then click again to expand it.

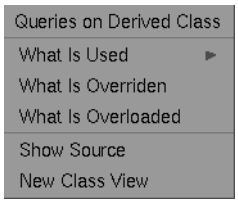


Figure 6-19 Queries on Derived Class

Now select the class MenuWindow by clicking on it with the left mouse button, then hold down the right mouse button anywhere in the related class list to bring up the Queries on Derived Class pop-up menu (see Figure 6-19).

Try finding the members of the current class that are overridden by choosing "What Is Overridden" from the menu, and examine the results in both the Class View and the Static Analyzer windows. (The other items won't find results in this example.)

Making Queries on Classes That the Current Class Uses

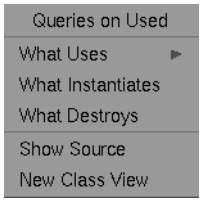


Figure 6-20 Queries on Used "What Uses"

In this section you'll examine a class that the current class uses. Scroll until the heading ***USES*** is visible in the related class list, and select the class Application using the left mouse button. Hold down the right mouse button anywhere in the related class list to bring up the Queries on Used pop-up menu.

Try finding the methods that call members of Application by selecting "By Calling Methods" from the "What Uses" submenu (see Figure 6-20). Be sure to check the results in the Static Analyzer window as well.

Making Queries on Classes That the Current Class is Used By

In this section you'll examine a class that uses the current class. For this example, we're going to change to a different current class, the class Application. Instead of going back to the current class text field, however, we're going to use a shortcut.

1. Double-click on any mention of Application in the related class list (the currently selected one is fine). Notice that the current class in the Class View window switches to Application, and that the member and related class lists are updated accordingly. You can change classes from the related class list at any time by double-clicking on the class you wish to make current.

2. Scroll until the heading ***USED BY*** is visible in the related class list, and select the class BouncingBall using the left mouse button. Hold down the right mouse button anywhere in the related class list to bring up the Queries on Users pop-up menu as shown in Figure 6-21.

   This menu is very similar to the Queries on Used menu from the last section. Try finding the members of Application that BouncingBall calls by selecting "By Calling Methods" from the "What Is Used" submenu.

   Remember that the results for the next query appear only in the Static Analyzer window.

3. To see the results of this query, you must have the *Show in Static Analyzer* toggle set, or you must click on the *Last Query* button after making the query. Now, double-click on BouncingBall to make it the new current class. Select the class AddBallCmd under the heading ***USED BY***. Find the members of AddBallCmd that instantiate BouncingBall by selecting "What Instantiates" from the Queries on Users pop-up menu (see Figure 6-22).

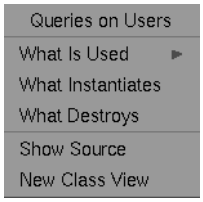


Figure 6-21 Queries on Users "What is Used"

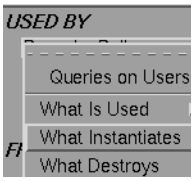


Figure 6-22 Queries on Users "What Instantiates"

Returning to a Previous Current Class

Now let's return to the previously displayed current class, Application. There's an easy way to do this: go to the History menu in the Class View menu bar, and choose "Show Previous Class," as shown in Figure 6-23. The current class reverts to whatever it was before the last change.

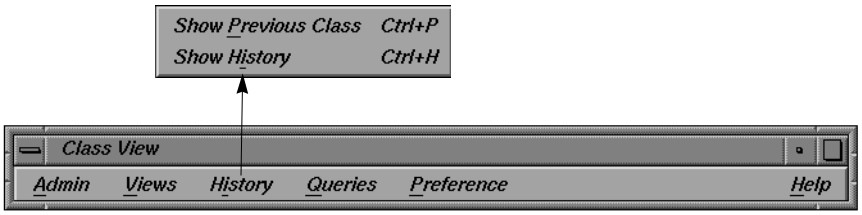


Figure 6-23 Using History Menu to Show Previous Class

Now try the other item from the History menu, “Show History.” This item brings up the List of Classes Shown window (see Figure 6-24) that lists each of the previous current classes you have examined, in the order you examined them. To choose one, double click on the name in the list. For now, though, click on the *Cancel* button.

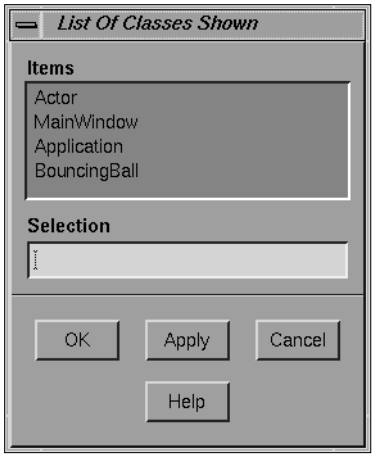


Figure 6-24 “Show History” Opens List of Classes Shown Window

Making Queries on Friends

Next you’ll examine the kinds of friend relations that Class View supports. For a selected friend function (of the current class Application), you’ll find methods in the following choices:

- current class that the friend function uses
- current class that uses the friend class
- friend class that uses the current class

Finding Current Class Methods That a Friend Function Uses

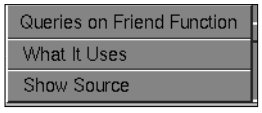


Figure 6-25 Queries on Friend Function

Locate the heading ***FRIEND FUNCTIONS*** in the related class list, and select the function listed under it. Hold down the right mouse button in the related class list to bring up the Queries on Friend Function pop-up menu (see Figure 6-25). If you try "What It Uses," you'll find that the function **`main()`** uses no members of the current class. However, double-clicking the function opens the related Source View.

Finding Current Class Methods That Use a Friend Class

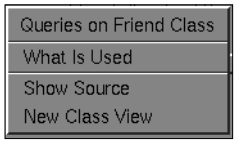


Figure 6-26 Queries on Friend Class

Now locate the heading ***FRIENDS*** in the related class list, and select the class listed under it. Hold down the right mouse button in the related class list to bring up the Queries on Friend Class pop-up menu (see Figure 6-26). Try the query item "What Is Used." Check the results in both the Class View and Static Analyzer Windows to see if it highlights all members of the current class that the selected friend class uses.

Finding Friend Class Methods That Use the Current Class

Finally, locate the heading ***FRIEND OF*** in the related class list, and select the class listed under it. Hold down the right mouse button in the related class list to bring up the Queries on Friend Of Class pop-up menu (see Figure 6-27).

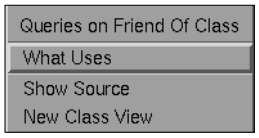


Figure 6-27 Queries on Friend of Class

Try the query item "What Uses." Check the results in both the Class View and Static Analyzer Windows to see if it highlights all members of the current class that use the friend class.

Now that you're familiar with the basic functions available within the Class View window, let's look at the C++ Browser's other two main windows, the Class Graph window and the Call Graph window.

You can minimize (but don't exit) the Static Analyzer window; you won't be using it for the rest of the session. (If you exit the Static Analyzer, you also exit the Browser.)

Using the Class Graph Window

The Class Graph window provides you with a graphical view of your fileset's class hierarchy. Open the Class Graph from the Views menu in the Class View window. The Class Graph displays the complete class hierarchy as found in the fileset, according to one of four available relationships, and highlights the current class. From there, you can prune the graph of classes until you are viewing only those classes that you are interested in.

The Class Graph has the same basic user interface as the other graph views found in the WorkShop suite of tools, which are explained in Appendix A, "Graphical Views in the CASEVision Environment," in the *CASEVision/WorkShop User's Guide*. For functions specific to the C++ Browser Class Graph window, refer to "Class Graph and Call Graph Displays" on page 117.

Opening a Class Graph Window

Go to the Views menu in your Class View window and you'll see the selections shown in Figure 6-28.

Choose "Show Inheritance Graph." The Class Graph window appears (it takes a little bit of time for the browser to access the static analysis database). You can resize the Class Graph by dragging the window frame with the left mouse button and by using the *zoom in* and *zoom out* buttons. This graph represents the complete inheritance structure of the example application. Later on, you'll see how to change the relationship being viewed from within the Class Graph window.

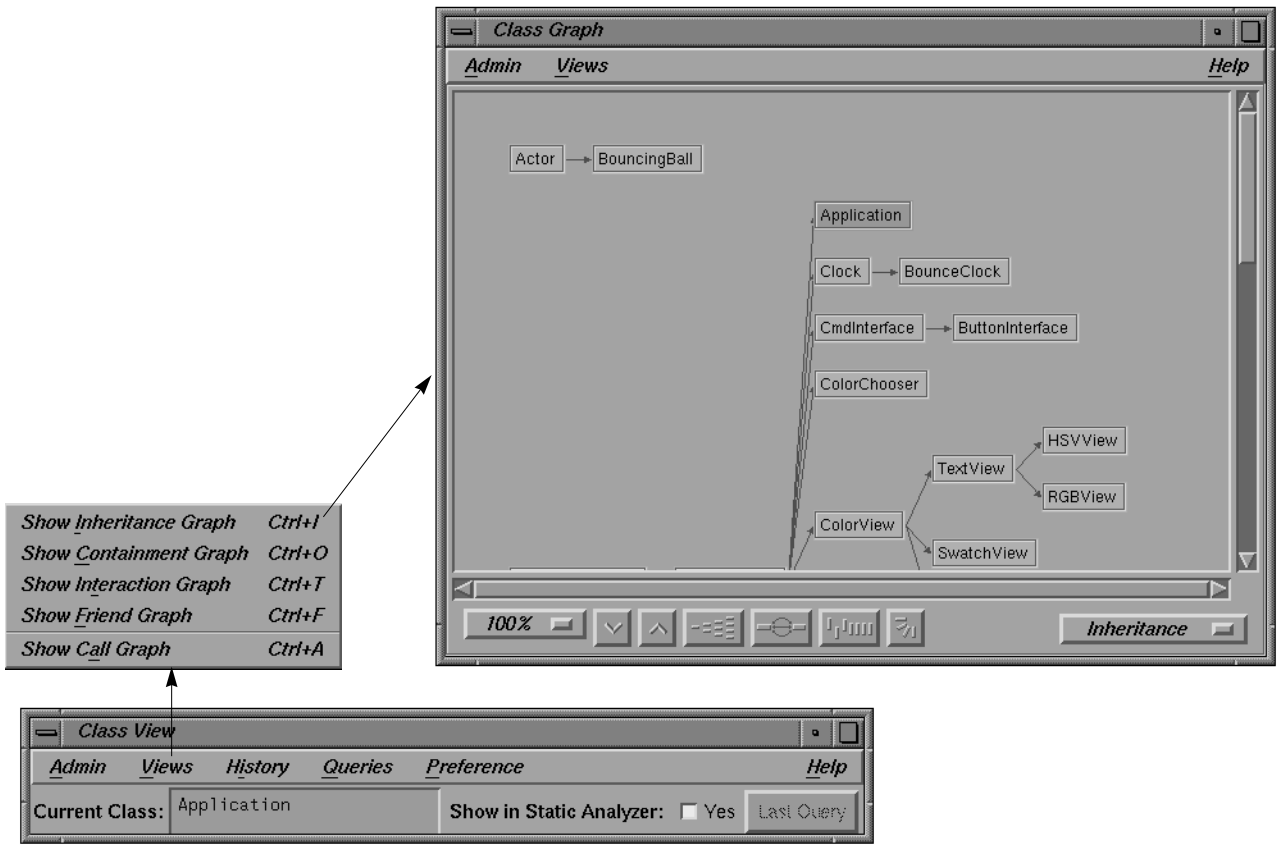


Figure 6-28 Inheritance Class Graph

Pruning the Class Graph View

Once you are comfortable with the Class Graph window, scroll until you find the class Application. This is the current class in your Class View window, and is highlighted in the Class Graph window.

You may have noticed that the graph is large and somewhat complicated, and this is only a small example! Luckily, there's an easy way to prune the classes displayed so that only those related to the current class are displayed.

1. Go to the View menu in the Class Graph window, and select “Show All Related.” The graph changes so that only the current class (Application) and its direct relatives (ancestors UIComponent and BasicComponent) are visible.
2. Go back to the Class View for a moment, and change the current class to Application’s parent, UIComponent. Notice that the Class Graph window updates to reflect the change. Note that the view is still restricted to those classes related to the current class: all ancestors and descendants.
3. Now choose “Show Butterfly” from the View menu. Now the graph is even more restricted, showing only immediate relations (in this case, parents and children).

Changing the Current Class From the Class Graph Window

Changing the current class from the Class View can be troublesome when you’re using the Class Graph. Luckily, you can change the current class from inside the Class Graph by double-clicking on a class node with the left mouse button. Choose “Show All” from the View menu. Find the class ColorView in the Class Graph window, and double-click to make it the current class. Note that the Class View window and the butterfly view in the Class Graph window both update accordingly.

Switching the Relationship Viewed by the Class Graph

Now you’ll change the kind of relationship you’re going to view. First, choose “Show All” from the View menu, just to make the graph more exciting. Then use the option menu in the lower right corner of the Class Graph window to choose a new relationship. Note the change in color of the arcs for the differing relationships. Figure 6-29 shows the Interaction relationships.

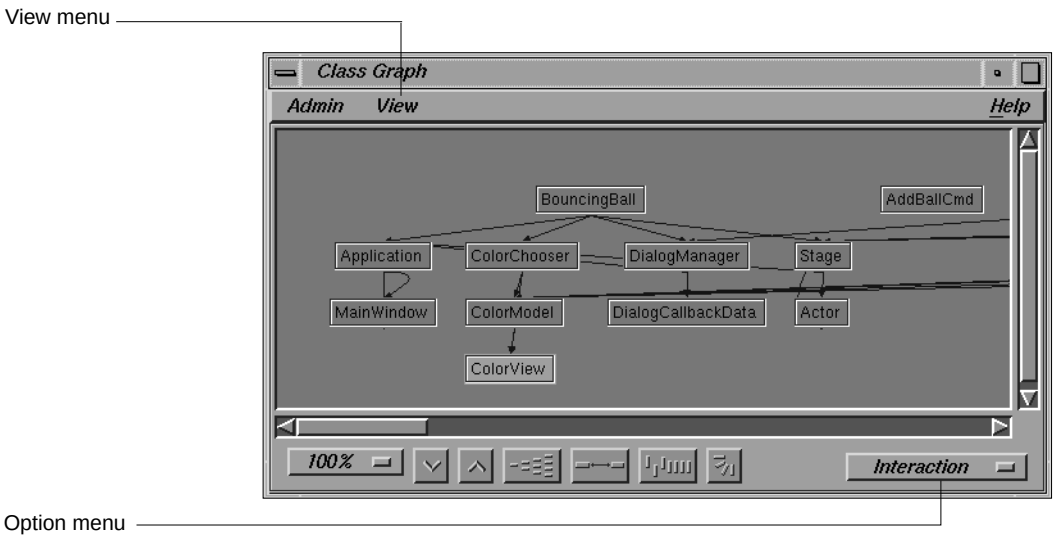


Figure 6-29 Interaction Relationship in Class Graph Window

Experiment with the different views and relationships available in the Class Graph window for as long as you like. When you're finished, close the Class Graph by choosing "Close" from the Admin menu.

Now you've seen the basic functions available within the Class Graph window. Your next exercise is to examine the Call Graph window.

Using the Call Graph Window

The Call Graph window provides you with a graphical view of the methods and their calls in your classes. Open the Call Graph from the Views menu in the Class View window, or the Queries on Methods pop-up menu for a selected method of the current class. The Call Graph displays the calling structure of any method you choose to add from the Class View window.

The Call Graph, like the Class Graph window, has the same basic user interface as the other graph views found in the WorkShop suite of tools. This interface is explained in the appendix titled “Graphical Views in the CASEVision Environment” in the *CASEVision/WorkShop User’s Guide* and are also covered in “Call Graph Window” on page 124. This section familiarizes you with the functions specific to the C++ Browser Call Graph window.

Opening a Call Graph Window

Go to the Views menu of your Class View window and select the first item, “Show Call Graph.” You’ll probably want to resize the window (even though there’s nothing in it yet) so that a comfortable area is visible.

Adding Methods

Now go back to the Class View window for a moment, and make ControlPanel the current class. Find and select the method named:

```
ControlPanel& ControlPanel(Widget,char*,Clock);
```

You may have to stretch out the Class View window and slide the sash (central divider) to see the whole name.

Once you’ve selected it, choose “Call Graph” from the Queries on Methods pop-up menu. Choose “Add” from the Call Graph submenu. The selected method and its calling structure are added to the Call Graph window as shown in Figure 6-30.

Note: If a Call Graph window is closed, you can open it by choosing “Call Graph”: “Add” from the Queries on Methods pop-up menu. This provides a method of the current class that is selected in the member list.

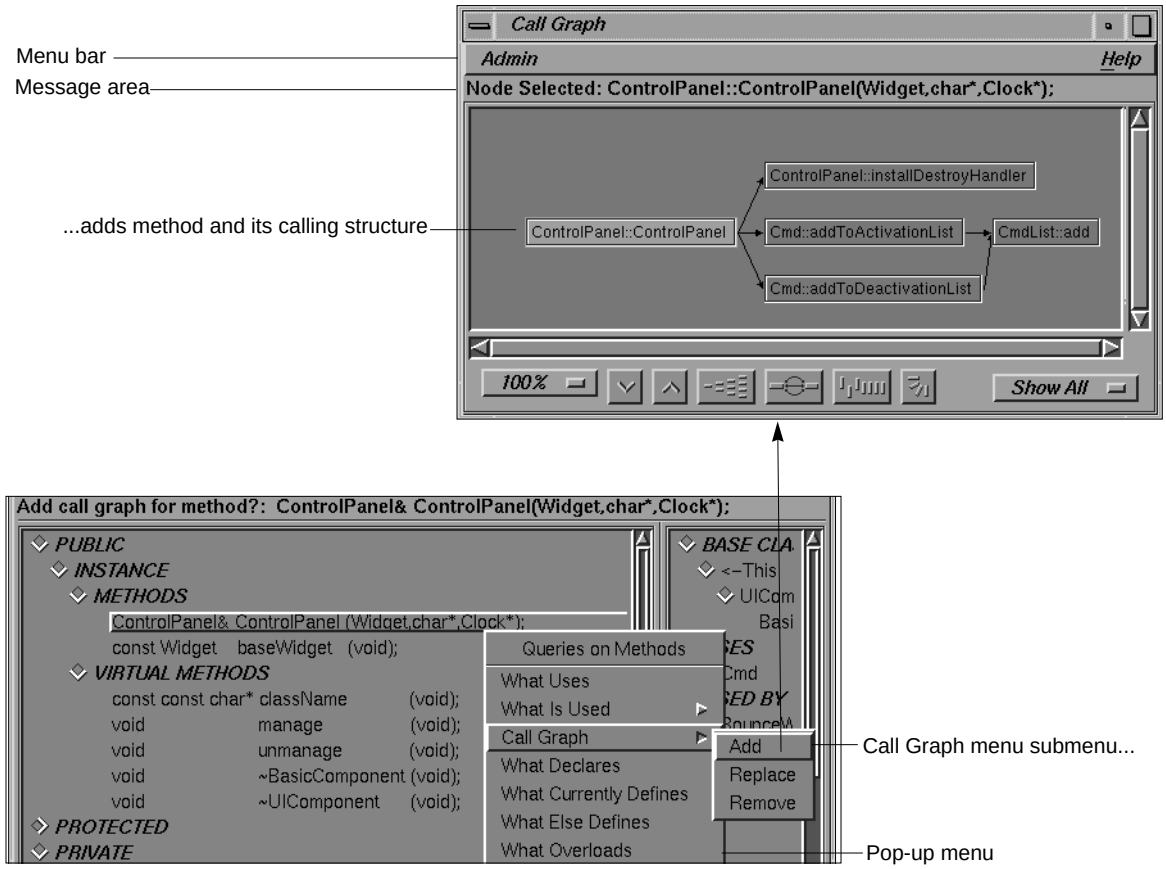


Figure 6-30 Adding a Method to the Call Graph Window

Showing Method Argument Lists

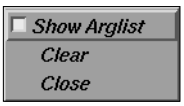


Figure 6-31 Toggling "Show Arg List"

The Call Graphnodes do not show argument lists, by default. Select a node of the calling structure. Look at the "Show Arg List" toggle from the Admin menu (see Figure 6-31). Turn it on by releasing the left mouse button directly over the toggle. Notice that each method in the Call Graph now shows its argument list. The node that you clicked matches the Node Selected information in the message area over the display.

Replacing Methods

Go back to the Class View. Find and select the method named:

```
const Widget baseWidget(void);
```

Select “Replace” from the Call Graph submenu of the Queries on Methods pop-up menu. Note that the Call Graph is cleared, then the newly selected method is added to the Call Graph. The graph shows where the method is currently defined. Here BasicComponent provides the current definition of baseWidget.

Add some other methods to the Call Graph. To replace part of the display, use “Remove” and “Add” from the Call Graph submenu of the “Queries on Methods” pop-up menu, with selected individual methods in the member list.

Viewing Method Source

Double-click on any method node displayed in the Call Graph window. A Source View appears, highlighting the position of the method’s source code.

Generating Reference (Man) Pages

The Browser generates reference page templates from your classes so that all you have to do is fill in the descriptions and provide comments. To create reference pages for classes in the fileset, follow these steps:

1. Select “Generate Man Pages” from the Class View Admin menu. The Generate Man Pages window opens, as shown in Figure 6-32.

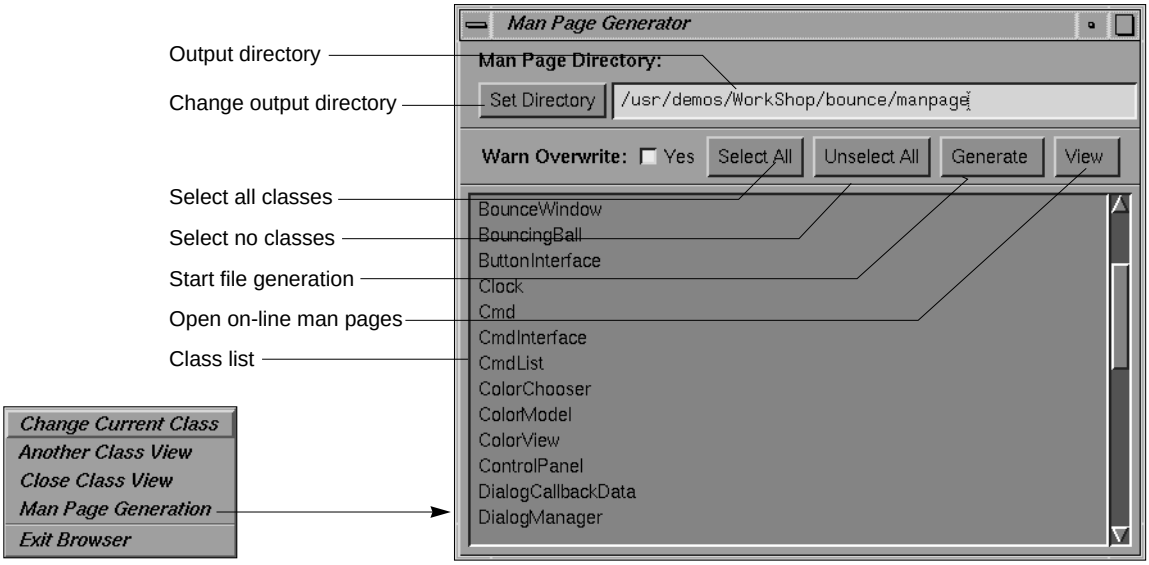


Figure 6-32 Generating Man Pages

2. Output files go in the directory shown in the *Man Page Directory* field. If you would like to specify a different output directory (an existing directory, where you have write permission), click *Set Directory*. The Select Man Page Directory dialog box lets you specify your choice. Click *OK* or *Cancel* to close the window.
3. Select classes from the class list. Use *Select All* to select every class in the source directory. To select no classes, click *Unselect All*. When you are satisfied with the classes selected in the class list, go on to the next step.
4. Click *Generate*. Wait for a few seconds while your files are generated.
5. To view the output files, Click *View*. A *winout* window opens as shown in Figure 6-33. You can edit the files using any text editor, such as *vi*.

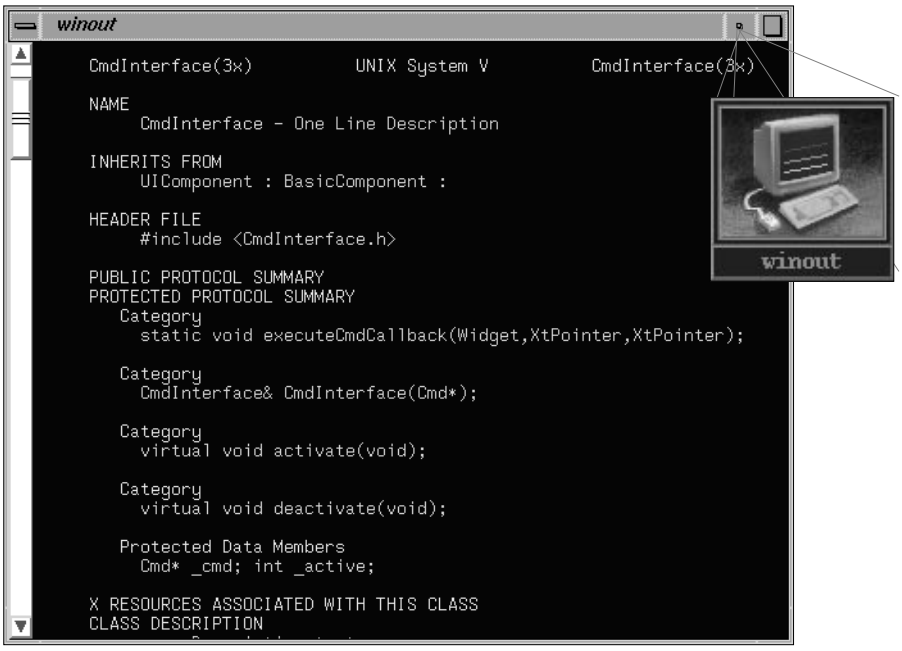


Figure 6-33 Man Page Template

Ending the Session

You've reached the end of the sample session. You can exit both the Static Analyzer and the C++ Browser by choosing "Exit" from the Static Analyzer's Admin menu.

Chapter 7

C++ Browser Reference

This chapter describes in detail the function of each window, menu, keyboard accelerator, and display in the C++ Browser user interface. For a task-oriented description of commonly used functions, refer to Chapter 6, “Using the C++ Browser: A Sample Session.”

This chapter contains the following sections:

- “Class View Window”
- “Class Graph and Call Graph Displays”
- “Class Graph Window”
- “Call Graph Window”
- “Customizing the C++ Browser”

Class View Window

Class View is the primary C++ Browser window (see Figure 7-1). It opens when you select “C++ Browser” from the Admin menu of the WorkShop Static Analyzer. Class View displays class members and related classes of a selected class, called the *current class*. Class View lets you perform a variety of static analysis database queries using the display. Detailed query results can be displayed in Source Views and in the Static Analyzer. You can launch graphical views of classes or calls from the Class View window to enhance the information. Also, you can generate reference pages from Class View.

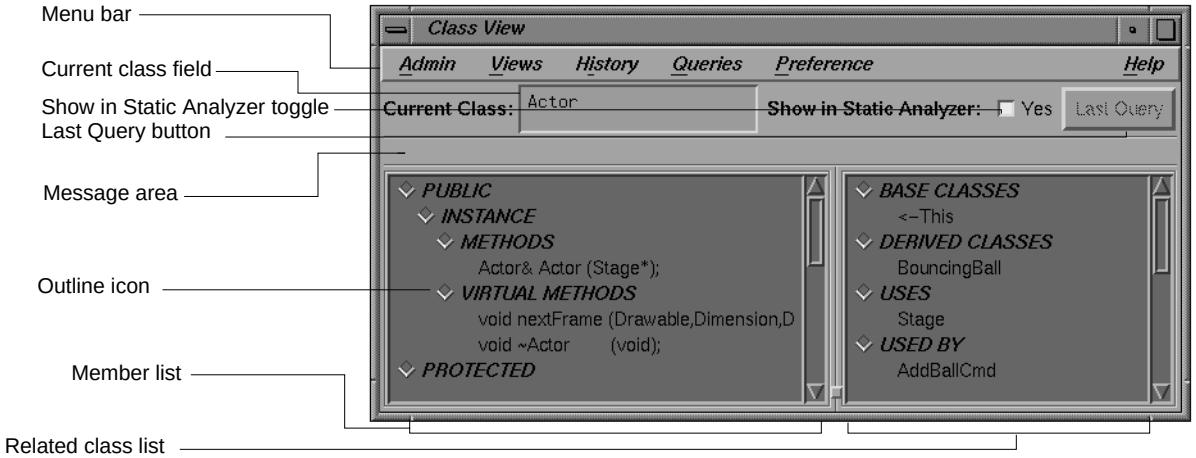


Figure 7-1 Class View Window Elements

The next sections describe the function of each element of the Class View window.

Class View Menu Bar

This section describes the menus found in the menu bar of the Class View window (see Figure 7-2). By choosing the dashed line (the first item in each of the menus), you can “tear off” the menu from the menu bar, so that it is displayed in its own window, and is thus always visible. Some menus contain submenus, which can also be torn off and displayed in separate windows. Finally, this section provides an alphabetical lookup table of the keyboard accelerators that are shortcuts for many of the menu selections.

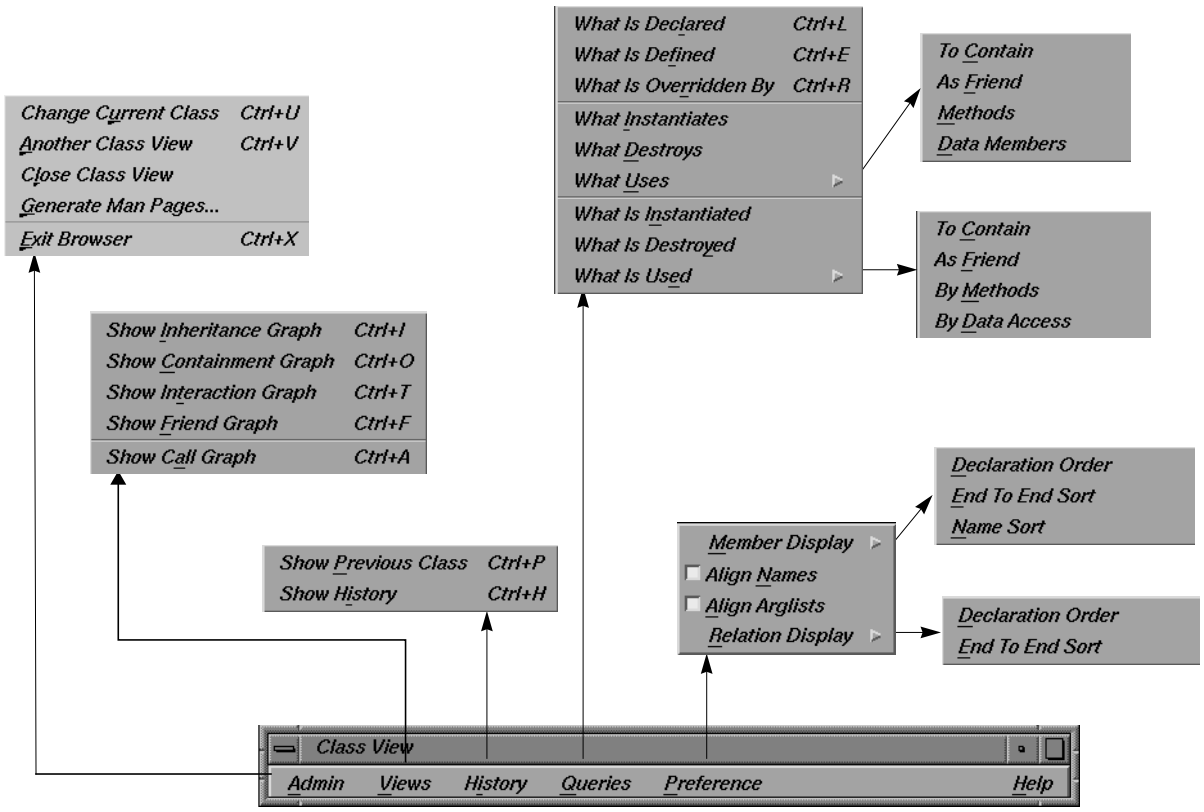


Figure 7-2 Class View Menu Bar

Admin Menu

The Admin menu contains commands for selecting a new current class, for manipulating Class View windows, generating reference pages (man pages), and exiting the Class View.

"Change Current Class"
: lets you select a new current class without manually typing it into the current class text field. Choosing this command

opens a class chooser window (List Of Classes, see Figure 7-3) that contains a scrolling list of all the classes available from the current fileset. To select a class, choose one from the list by clicking on it with the left mouse button and then pressing the *Accept* button. To select a class and simultaneously close the chooser window, click on the item and then click on the *OK* button, or simply double-click on the item in the list. The *Cancel* button closes the chooser window without changing the current class.

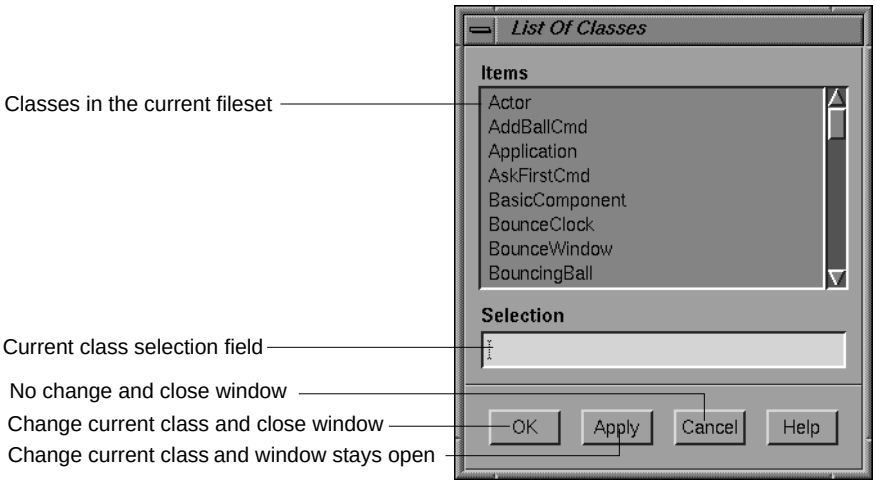


Figure 7-3 List of Classes Chooser Window

"Another Class View"
: creates an identical copy of the Class View window. All current information displayed within the initial window is preserved by the copy, but connections to Class Graph and Call Graph windows are not carried over to the new Class View window.

"Close Class View"
: shuts the Class View window from which the command was selected. Also, any associated windows, such as Graph or List of Classes, are shut.

“Generate Man Pages...”

opens a Man Page Generator window that lets you create reference page templates for classes (see Figure 7-4). For a tutorial about using this feature, refer to “Generating Reference (Man) Pages” on page 89.

Select individual classes by clicking on them. To start over with no classes selected, click *Unselect All*. If you want a reference page for every class in the list, click *Select All*. Clicking the *Generate* button creates a reference page template for each selected class. If reference pages exist for selected classes, the browser warns you, unless you toggle *Warn Overwrite:* to No.

Output files go in the directory shown in the *Man Page Directory* field, if it exists. To specify a different output directory using the Select Man Page Directory dialog box, click the *Set Directory* button in the Man Page Generator window.

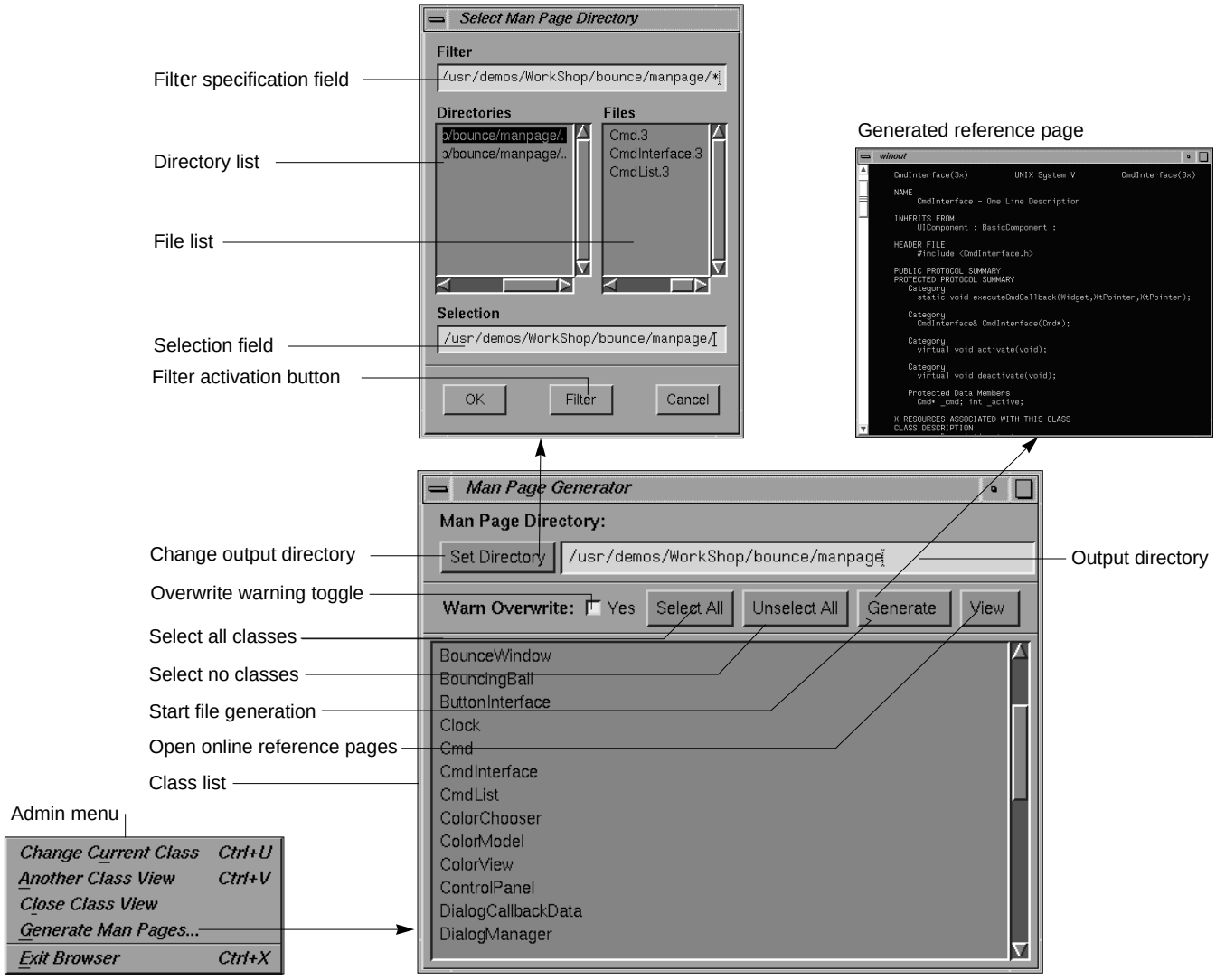


Figure 7-4 Reference Page Generation

"Exit Browser" quits the C++ Browser, closing all windows launched from it (except Source View). The Static Analyzer window from which the browser was launched is not affected.

Views Menu

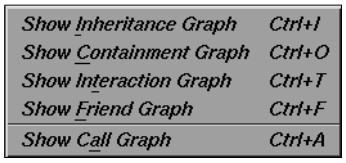


Figure 7-5 Views Menu

The Views menu contains commands for opening windows with graphical views, as shown in Figure 7-5. For descriptions of the display and controls, refer to “Class Graph and Call Graph Displays” on page 117. Each of the first four selections opens a Class Graph window for the current class, with a specific relationship; refer to “Class Graph Window” on page 122. The last selection opens a Call Graph window; refer to “Call Graph Window” on page 124.

“Show Inheritance Graph”
: describes the relationship between base classes and derived classes.

“Show Containment Graph”
: describes the relationship of container classes to the classes they use as components.

“Show Interaction Graph”
: describes the relationship of used classes to the classes that are their users.

“Show Friend Graph”
: describes the relationship of classes declaring friends to the classes they declare.

“Show Call Graph”
: opens a Call Graph window. To operate on it, choose a current class from the “List Of Classes” chooser window. Then select a method from the member list display (left list on the Class View window). Press right mouse button to pop up the “Queries on Methods” menu and choose one of the commands in the “Call Graph” submenu. This menu provides a more selective view than is available in the Static Analyzer.

History Menu

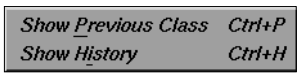


Figure 7-6 History Menu

The History menu contains commands that let you quickly select previously chosen classes for display in the Class View window (see Figure 7-6). If no class was selected previously, a message appears.

"Show Previous Class"

sets the current class to the previously displayed class and the information in the Class View window changes to reflect this.

"Show History"

opens a "List of Classes Shown" chooser window (see Figure 7-7) that presents the list of previously displayed current classes in chronological order, with the most recently displayed class at the bottom of the list.

To select a class, choose one from the list by clicking on it with the left mouse button, and then press the *Apply* button. To select a class and simultaneously close the chooser window, click on the item and then click on the *OK* button, or simply double-click on the item in the list. The selected class then becomes the current class, and the information in the Class View window changes to reflect this. The *Cancel* button closes the chooser window without changing the current class.

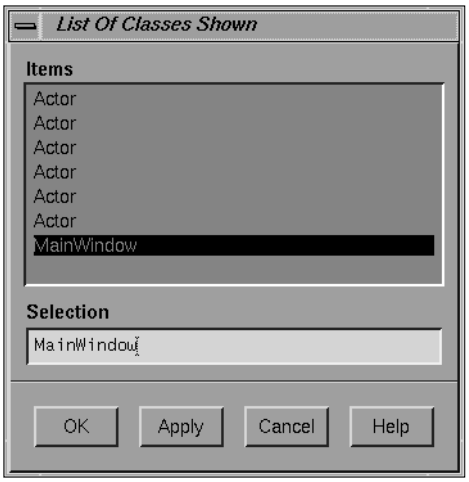


Figure 7-7 List of Classes Shown

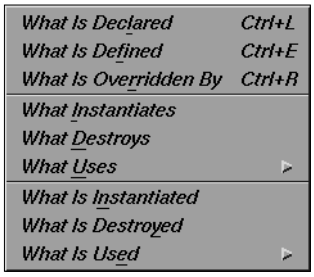


Figure 7-8 Queries Menu

Queries Menu

The Queries menu contains a set of predefined searches for information about the current class, as shown in Figure 7-8. The Class View window outlines display the results of each query by highlighting classes or members. If the *Show in Static Analyzer* toggle is set to *yes*, detailed results of the query appear in the WorkShop Static Analyzer window from which you launched the C++ Browser (refer to “Show in Static Analyzer Toggle” on page 104). Descriptions of the current query and the magnitude of the results of the query are shown in the message area between the current class field and the outline lists. The selections are:

“What Is Declared”
: displays all methods declared by the current class.

“What Is Defined”
: displays all members defined by the current class.

“What Is Overridden By”
: displays all inherited methods that the current class overrides.

“What Instantiates”
: displays classes that instantiate the current class by invoking its constructors or by using its new methods.

“What Destroys”
: displays classes that destroy the current class by invoking its destructors or by using its delete methods.

”What Uses” submenu (see Figure 7-9)
: queries which classes use the current class in these contexts:

- “To Contain” displays classes that use the current class as either an embedded or linked component.
- “As Friend” displays classes that use the current class as a friend class.
- “Methods” displays classes that use the methods defined by the current class.
- “Data Members” displays classes that use (by modifying, reading, or taking the address) data members defined by the current class.

Figure 7-9 “What Uses” Submenu of Queries Menu

"What Is Instantiated"
displays classes that the current class instantiates by invoking its constructors.

"What Is Destroyed"
displays classes that the current class destroys by invoking its destructors.

"What Is Used" submenu (see Figure 7-10)
queries which classes are used by the current class in these contexts:

- "To Contain" highlights classes that the current class uses either as embedded or linked components.
- "As Friend" highlights classes that the current class uses as friend classes.
- "By Methods" highlights classes whose methods are used by the current class.
- "By Data Access" highlights classes whose data members are assigned, read, or have their address taken by the current class.

Additional queries on classes, data members and methods are accessible from pop-up menus described in "Member List" on page 107 and "Related Class List" on page 111.

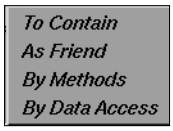


Figure 7-10 "What Is Used" Submenu of Queries Menu

Preference Menu

The Preference menu allows you to control how the class information is displayed in the window (see Figure 7-11). The selections are:

"Member Display" submenu (see Figure 7-12)
allows you to control how the class members are displayed. There are three choices:

- "Declaration Order" displays the members in order of their declaration.
- "End To End Sort" performs an end-to-end sort of the member display strings and displays the result.
- "Name Sort" performs a sort based on the name of the members and displays the result.

Figure 7-11 Preference Menu

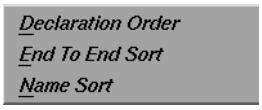


Figure 7-12 "Member Display" Submenu of Preference Menu

Figure 7-13 "Relation Display" Submenu of Preference Menu

"Align Names" aligns the member names in the display. A radio button indicates if this feature is enabled or disabled.

"Align Arglists" aligns the member function argument lists in the display. A radio button indicates if this feature is enabled or disabled.

"Relation Display" submenu (see Figure 7-13)
allows you to control how the class relations are displayed. There are two choices:

- "Declaration Order" displays the related classes in order of their declaration or the detection of their relation.
- "End To End Sort" displays a sorted list of related classes.

Help Menu

Figure 7-14 Help Menu

The Help menu contains commands that allow you to access online information and documentation for the C++ Browser, as shown in Figure 7-14. It is part of the Static Analyzer help system. See Chapter 2, "Using Online Help," in the *CASEVision Environment Guide* for more detailed information on how to use the C++ Browser help features.

"On Version" opens a window containing version number information for the Static Analyzer.

"On Window" invokes the CASEVision Help Viewer, which displays a descriptive overview of the current window or view and its graphical user interface.

"On Context" invokes context-sensitive help. When you select the "On Context" command, the normal mouse cursor (an arrow) is replaced with a question mark. When you click on graphical features of the application with the left mouse or position the cursor over the feature and press the **<F1>** key, the Help Viewer displays information on that context.

"Index..." invokes the Help Viewer, which displays the list of available help topics. You can browse alphabetically, hierarchically, or graphically.

Figure 7-15 Current Class Field

Current Class Field

The *Current Class* text field is directly below the menu bar (see Figure 7-15). It displays the name of the currently selected class in the Class View window. You can type into this field to change the current class.

Type a complete class name into the text field and then press the **<Enter>** key to make the change effective. If you type a partial string and then press the **<space bar>**, the browser attempts to complete the class name with a class from the fileset. It gives a beep if it finds more than one matching class name. If a match is made, press the **<Enter>** key to make the change effective.

If you type a question mark (?) into the Current Class field, a class chooser dialog window (List of Classes) opens. You can select a new class by left-clicking on a class name in the chooser window's scrolling list, or click once with the left mouse button and then click on the *OK* or *Apply* buttons. Clicking *Apply* leaves the chooser window open; other combinations close the chooser window.

Figure 7-16 *Show in Static Analyzer* Toggle

Show in Static Analyzer Toggle

The *Show in Static Analyzer* toggle is directly to the right of the *Current Class* field (see Figure 7-16). When the toggle is set (highlighted and labeled *Yes*), the results of all queries are displayed in the WorkShop Static Analyzer window from which the C++ Browser was launched, providing detailed source information not listed in the Class View window. If no results are found, and the Static Analyzer window is open, it comes to the front with an error message. Refer to the *CASEVision/WorkShop User's Guide* for more information on the Static Analyzer.

Figure 7-17 Last Query Button

Last Query Button

The *Last Query* button is to the far right, directly beneath the Help menu (see Figure 7-17). Clicking on this button displays the results of the most recent query in the WorkShop Static Analyzer window from which the C++ Browser was launched.

Class View Message Area

The Class View message area is located directly below the Current Class field (see Figure 7-18). This area displays the most recent query as an English sentence, listing both the query question and the name of the data member, method, or class that is the object of the query. The end of the line displays query results, such as how many members and classes are involved.

Class View Outline Lists

The lower two-thirds of the Class View window contains two side-by-side lists that contain information about the currently selected class in outline form (see Figure 7-18).

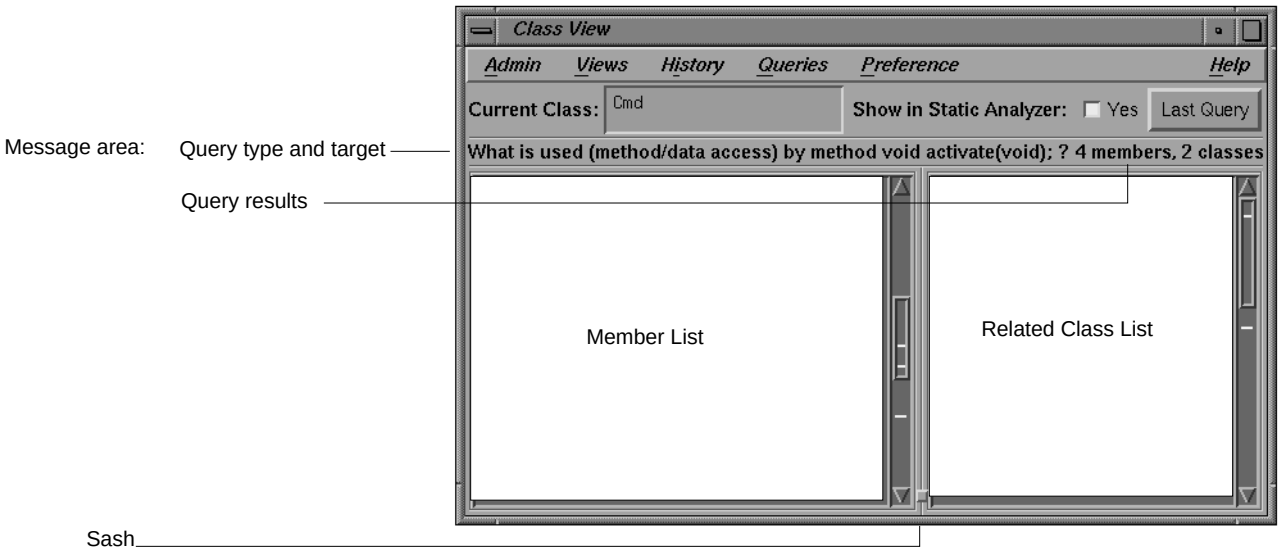


Figure 7-18 Class View Message Area and Outline Lists

- The lefthand, or *member* list, provides a detailed view of the members of the current class.
- The righthand, or related class list, displays the relationships of other classes and of friend functions to the current class.

Figure 7-19 Expanded List, Downward Pointing Icon

Figure 7-20 Collapsed List, Right-Pointing Icon

The organization of each of these lists is described in “Member List” on page 107 and “Related Class List” on page 111. This section describes features common to both lists. You can change the widths of the lists by moving the sash or central divider.

Outline Icons

The lists of members and class are organized in an outline format. By clicking on the outline icon to the left of a heading, you can collapse or expand the list under that category. The direction in which the outline icon points indicates if the heading is expanded or collapsed; downward-pointing outline icon indicates that the list is expanded (see Figure 7-19), while a right-pointing icon indicates that the category is collapsed (see Figure 7-19).

To collapse an outline list, click on the heading’s downward-pointing outline icon. Class View hides the entries for that heading, and the icon toggles to become a right-pointing arrow, as shown in Figure 7-21.

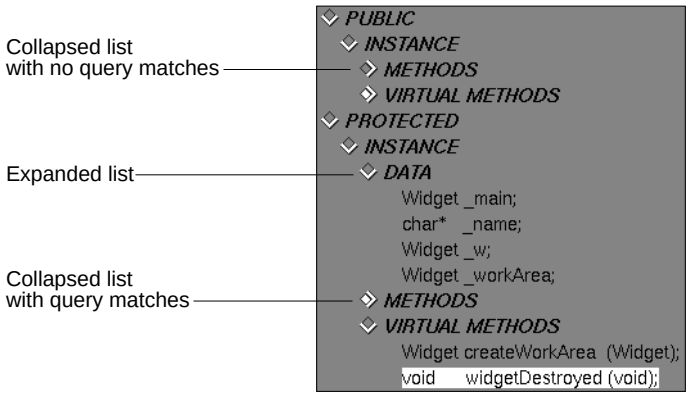


Figure 7-21 Outline Lists and Icons

To expand a list of members or classes under a particular heading, click on the heading’s right-pointing icon. Class View displays the entries for that heading, and the icon toggles to become a downward-pointing arrow.

Figure 7-22 Query Result in List, Highlighted Icon

A *highlighted* right-pointing outline icon indicates that one or more of the collapsed entries matches the most recent query (see Figure 7-22). Click on the outline icon to display the entry.

Annotated Scroll Bars and Highlighted Entries

Both outline lists make use of annotated scroll bars as a means of locating highlighted list entries. When you make a query on an entry in the member or related class list, all members or classes that satisfy the query are highlighted in their respective lists; if that entry is collapsed, the corresponding outline icon of the nearest exposed heading that contains the entry is highlighted instead. The annotated scroll bars show where highlighted entries occur in the scrolling list by displaying ticks in the highlight color at the proper location in the scroll bar. When the thumb of the scroll bar overlaps a given tick, the corresponding entry is visible in the list window.

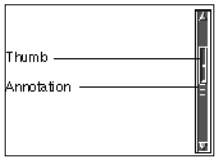


Figure 7-23 Annotated Scroll Bars

Member List

The Class View member list contains information on types, data members, methods, and virtual methods included in the current class.

Double-clicking on any member in the member list opens a Source View window for that member's code with the member's declaration highlighted. See Figure 7-24.

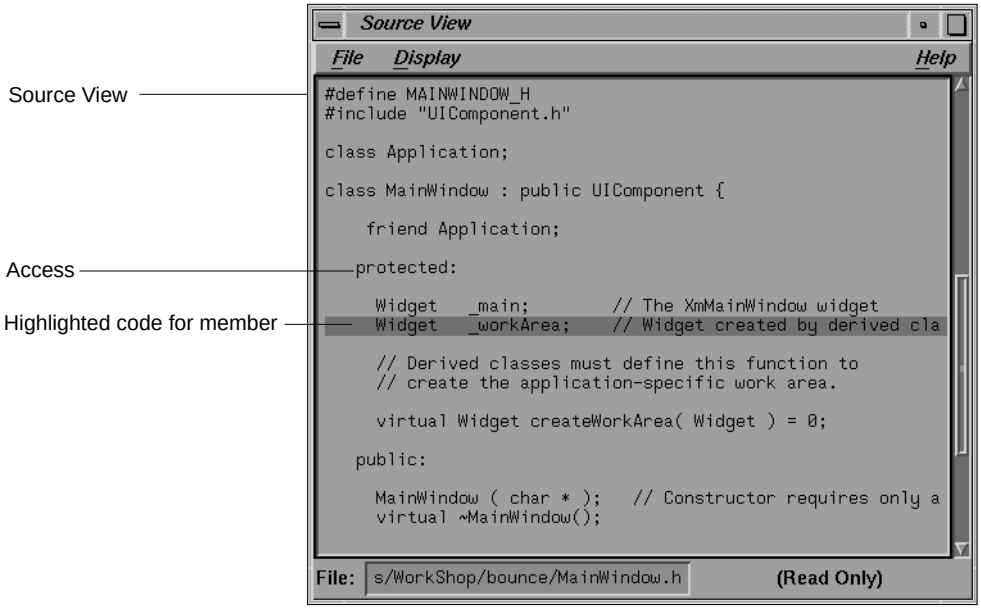


Figure 7-24 Source View of Class Data Member

Member List Structure

The member list sorts the members of the current class recursively into three nested lists according to the *access* specification (public, protected, private) of each member. Within each of the three resulting lists, the members are sorted once again by *scope* into two sublists (instance and static). Finally, within each of these sublists, members are displayed by member category type in this order: type members, data members, methods (member functions), and virtual methods.

Here is a schematic of the outline format for each nested list:

```
Access (Public, Protected, or Private)
   Scope (Instance or Static)
      Types
      Data
      Methods
      Virtual Methods
```

For a discussion of these concepts, refer to “C++ Class Structure and the Current Class” on page 55. The list organization is customizable. For more information, refer to “Customizing the C++ Browser” on page 128.

Member List Query Menus

This section details each of the pop-up query menus available from the member list display. To execute a query from one of these menus, select the member entry you wish to query on by left-clicking on it. Then, click and hold down the right mouse button anywhere in the member list display to open the corresponding query menu. By choosing the dashed line (the first item in each of the menus), you can “tear off” the menu from the menu bar, so that it is displayed in its own window, and is thus always visible.

The Queries on Data Members pop-up query menu performs these queries on selected data members:

“What Modifies”
: highlights all methods and classes in which the selected data member is assigned a value.

“What Reads”
: highlights all methods and classes in which the selected data member is read.

“What Accesses”
: highlights all classes where the selected data member is assigned a value, read, or its address is taken.

“What Defines”
: highlights the class that defines the selected data member.

The Queries on Methods pop-up query menu performs these queries on selected methods:

“What Uses”
: highlights all methods and classes that use the currently selected method.

“What Is Used” submenu
: contains these menu items:

  - “All (method and data access)” highlights all data members, methods, and classes the currently selected method uses.

- “Method Calls” highlights all methods called by the currently selected method.
- “Data Access” highlights all data members that have been assigned, read, or had their address taken by the currently selected method.
- “Data Modification” highlights all data members assigned by the currently selected method.
- “Data Read” highlights all data members read by the currently selected method.

“Call Graph” submenu
: contains these menu items:

  - “Add” adds the currently selected method and its calling structure to the Call Graph window, if one is open. If not, “Add” opens a Call Graph window before adding the method.
  - “Replace” replaces all methods in the display with the selected method and its calling structure in the Call Graph window.
  - “Remove” removes the currently selected method and its calling structure from the Call Graph window.

“What Declares”
: highlights the class that declares the currently selected method.

“What Currently Defines”
: highlights the class that provides the current definition for the method.

“What Else Defines”
: highlighs all classes that define the currently selected method.

“What Overloads”
: highlights all methods and classes that overload the currently selected method.

Related Class List

The Class View window lists the current class and its related classes in the related class list. Within this list, the current class is displayed as follows:

```
<- This
```

This class refers to the class in the Current Class text field.

Double-clicking on any class listed in the related class list makes it the new current class. Double-clicking on a friend function brings up a Source view window highlighting the function's definition.

Related Class List Structure

The related class list display is composed of seven sublists (not all of which may be in use for a given class):

- ***BASE CLASSES*** contains the current class and its ancestors, listed hierarchically.
- ***DERIVED CLASSES*** contains descendants of the current class, listed hierarchically.
- ***USES*** contains classes that the current class uses (that is, instantiates, destroys, interacts with, or contains).
- ***USED BY*** contains classes that use the current class.
- ***FRIEND FUNCTIONS*** contains global functions declared as friends by the current class.
- ***FRIENDS*** contains classes that are declared as friends by the current class.
- ***FRIEND OF*** contains classes that declare the current class as a friend.

The Base Classes sublist shows the ancestors of the current class, if any. Each indented class is an ancestor of the class listed above it. The Base Classes sublist indicates a *multiple inheritance* relationship by indenting the two or more parent classes to the same level. If a given class has ancestors, it is accompanied by an outline icon, which works in a similar manner to the outline icons in the member list. Each ancestor name is followed by its inheritance access type (public, protected or private) listed in parentheses.

This schematic gives an example of a possible Base Classes sublist:

```
BASE CLASSES
    <-This
        first_parent_of_This (access type)
                parent_of_first_parent_class (access type)
        second_parent_of_This (access type)
                parent_of_second_parent_class (access type)
```

The Derived Classes sublist shows the descendants of the current class, if any. Each indented class is a descendant of the class listed above it. If a given class has descendants, it is accompanied by an outline icon, which also works in a similar manner to the outline icons in the base classes sublist and member list. (See Figure 7-25 for an example.)

This schematic gives an example of a possible Derived Classes sublist:

```
DERIVED CLASSES
    first_child_of_This
        child_of_first_child_class
    second_child_of_This
        child_of_second_child_class
```

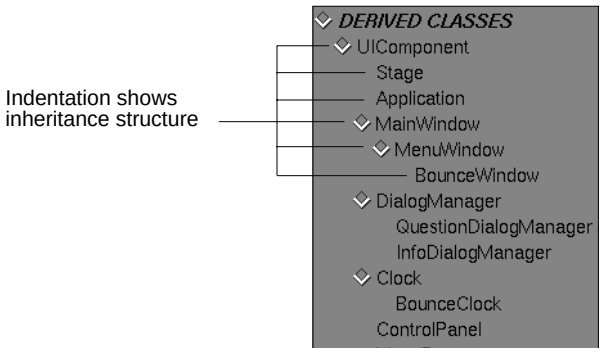


Figure 7-25 Class View Derived Classes List

Related Class List Query Menus

This section details each of the pop-up query menus available from the related class list display. To execute a query from one of these menus, select the name of the class you wish to query by clicking on it with the left mouse button. Then, click and hold down the right mouse button anywhere in the

related class list display to open the corresponding query menu. By choosing the dashed line (the first item in each of the menus), you can "tear off" the menu from the menu bar, so that it is displayed in its own window, and is thus always visible.

Most of the related class query menus have two generic commands (shown in Figure 7-26) that affect the selected item:

"Show Source" opens a Source View window on a file containing the declaration of the selected item. The first line of the declaration is highlighted in the source.

"New Class View"
opens a new Class View window displaying the selected class.

Figure 7-26 Queries on Current Class Pop-Up Menu

These two items are omitted from the lists below.

The Queries on Current Class pop-up query menu (see Figure 7-26) contains items identical with those in the Queries menu in the Class View menu bar. It also contains the two generic commands described previously.

The Queries on Base Class pop-up query menu, under the heading ***BASE CLASSES***, performs these queries on selected classes:

"What Is Declared"
highlights all methods declared by the selected base class.

"What Is Defined"
highlights all members defined by the selected base class.

"What Is Overridden"
highlights all methods belonging to the selected base class that are overridden by the current class.

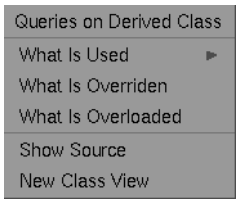


Figure 7-27 Queries on Derived Class Pop-Up Menu

Queries on Derived Class pop-up menu, under the heading ***DERIVED CLASSES*** (see Figure 7-27), allows you to perform functions on selected classes:

"What Is Used" submenu
contains these queries:

- "by Accessing Any Member" highlights all members (of the current class) that the selected derived class uses.
- "by Calling Methods" highlights all methods (of the current class) that the selected derived class uses.
- "by Accessing Data Members" highlights all data members (of the current class) that the selected derived class modifies, reads, or takes the address of.
- "by Modifying Data Members" highlights all data members (of the current class) to which the selected derived class assigns a value.
- "by Reading Data Members" highlights all data members (of the current class) from which the selected derived class reads a value.

"What Is Overridden"
: highlights all members of the current class that are overridden by the selected derived class.

"What Is Overloaded"
: highlights all members of the current class that are overloaded by the selected derived class.

The Queries on Used pop-up menu, under the heading ***USES*** (see Figure 7-28), allow you to perform these functions on selected classes:

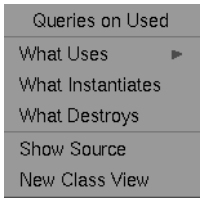


Figure 7-28 Queries on Used Pop-Up Menu

"What Uses" submenu
: contains these queries:

- "by Accessing Any Member" highlights all members (of the current class) that use the selected class.
- "by Calling Methods" highlights all methods (of the current class) that use the methods of the selected class.
- "by Accessing Data" highlights all data members (of the current class) that modify, read, or take the address of data members of the selected class.
- "by Modifying Data" highlights all data members (of the current class) that assign a value to data members of the selected class.

- "by Reading Data" highlights all data members (of the current class) that read a value from data members of the selected class.

"What Instantiates"
highlights all members of the current class that instantiate the selected class.

"What Destroys"
highlights all members of the current class that destroy the selected class.

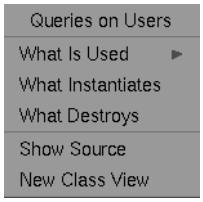


Figure 7-29 Queries on Users Pop-Up Menu

The Queries on Users pop-up query menu, under the heading ***USED BY*** (see Figure 7-29), performs these queries on selected classes:

"What Is Used" submenu
contains these queries:

- "by Accessing Any Member" highlights all members (of the current class) that the selected class uses.
- "by Calling Methods" highlights all methods (of the current class) that the selected class uses.
- "by Accessing Data" highlights all data members (of the current class) that the selected class reads, modifies, or takes the address of.
- "by Modifying Data" highlights all data members (of the current class) to which the selected class assigns a value.
- "by Reading Data" highlights all data members (of the current class) from which the selected class reads a value.

"What Instantiates"
finds all members of the selected class that instantiate the current class and displays them in the Workshop Static Analyzer window. To see the results of this query, you must have the *Show in Static Analyzer* toggle set, or you must click on the *Last Query* button after making the query.

"What Destroys"
finds all members of the selected class that destroy the current class and displays them in the Workshop Static

Analyzer window. To see the results of this query, you must have the *Show in Static Analyzer* toggle set, or you must click on the *Last Query* button after making the query.

The Queries on Friend Function pop-up query menu is listed in the related class list for the sake of convenience, under the heading ***FRIEND FUNCTIONS***. Double-clicking a friend function opens a Source View window highlighting the first line of the function definition. This menu performs the query described below on the selected friend function.

"What It Uses" highlights all members of the current class that the selected friend function uses.

The Queries on Friend Class pop-up query menu, under the heading ***FRIENDS***, performs the query described below on a selected class.

"What Is Used" highlights all members of the current class that the selected friend class uses.

The Queries on Friend Of pop-up query menu, under the heading ***FRIEND OF***, performs the query describe below on a selected class.

"What Uses" highlights all members of the current class that use the friend class.

Keyboard Accelerators

To issue Class View commands directly from the keyboard, use the keyboard accelerators in Table 7-1. The accelerators are listed in alphabetical order.

Table 7-1 Keyboard Accelerators for Class View

Command	**Ctrl +** *key*
Change to previous class	P
Exit C++ Browser	X
Highlight what is declared by the current class	L
Highlight what is defined by the current class	E

Table 7-1 Keyboard Accelerators for Class View

Command	**Ctrl +** *key*
Highlight what is overridden by the current class	R
Open another Class View	V
View Call Graph	A
View Containment Class Graph	O
View Friend Class Graph	F
View Inheritance Class Graph	I
View Interaction Class Graph	T
View list of classes	U
View list of classes shown previously	H

Class Graph and Call Graph Displays

You invoke the Class Graph and Call Graph windows from the Views menu of the Class View window. These windows show graphical views of the current fileset. As their names suggest, a Class Graph displays class relationships, and a Call Graph displays methods and their calling relationships. However, the displays and controls have the same features.

Figure 7-30 shows the *bounce* fileset class hierarchy. The Class Graph Context View shows an overview of the selected class with respect to the fileset. This section describes in detail how to view classes in a Class or Call Graph window.

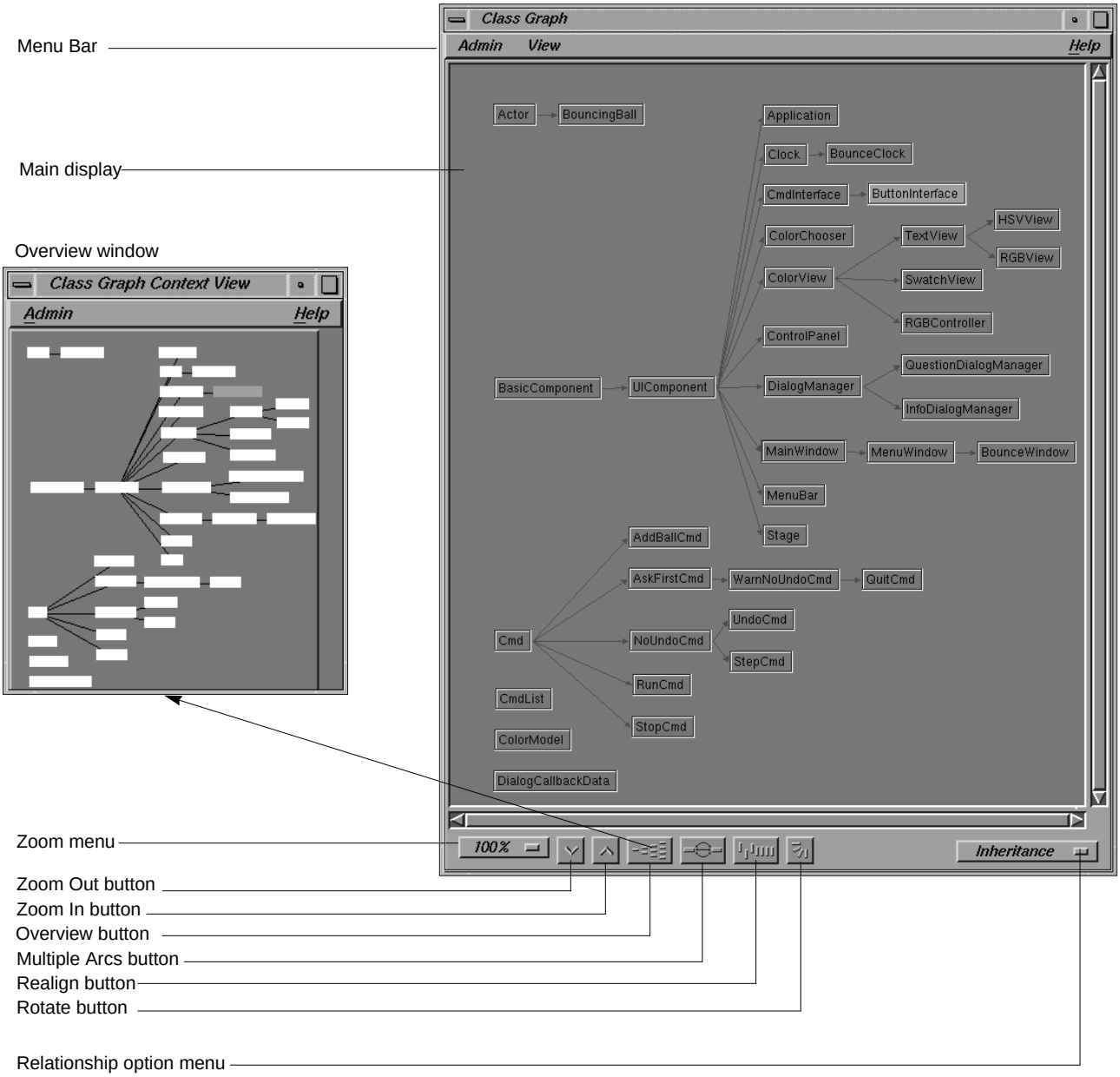


Figure 7-30 Class Graph with Context View—Bounce Hierarchy

Class and Call Graph Main Display

The Class and Call Graph main display presents a graphical view of classes (or class nodes) and methods (or method nodes) and their relations. These nodes can be selected, hidden, collapsed, expanded, and moved. Double-clicking nodes are documented under separate headings for each kind of window.

Selecting Nodes

There are several ways to select nodes:

- To select a single node, click on it with the left mouse button. It becomes highlighted in red.
- To select a node and all of its descendant nodes, hold down the **<Shift>** key while clicking on the desired node.
- To select an arbitrary group of nodes, hold down the left mouse button in an empty area of the display, and then drag it. A box is drawn; move the mouse so that all nodes you want to select are inside the box, then let go of the mouse button.

Node Pop-Up Menus

Two pop-up menus in the main display manipulate nodes: the Class or Method Node pop-up menu, and the Selected Node pop-up menu.

The Class or Method Node Pop-Up Menu appears if you hold down the right mouse button over a particular node. The node's name appears as the menu title. The node does not need to be selected for the menu commands to apply. It contains these commands:

- "Hide Node" hides the named node and leaves its descendants visible and disconnected from the rest of the graph.
- "Collapse Subgraph" hides the named node and its descendants.
- "Show Immediate Children" adds any non-visible children of the named node to the graph. If a node has hidden children, it contains an arrow within the node pointing to the right (see Figure 7-31).
- "Show Parents" adds any hidden parents of the named node to the graph.

Figure 7-31 Hidden Children

Any menu items that are not applicable for a given node are grayed out when the menu is opened.

The Selected Nodes pop-up Menu appears if you hold down the right mouse button over an empty portion of the main display. The commands apply to any selected nodes.

- “Hide Selected Nodes” hides the selected nodes, and leaves their descendants visible and disconnected from the rest of the graph.
- “Collapse Selected Nodes” hides the selected nodes and their descendants.
- “Expand Selected Nodes” adds all non-visible descendants of the selected nodes to the graph.

Moving Nodes

You can move any single node (selected or not) by holding the middle mouse button down over the desired node and dragging to the new desired position. You can move a group of nodes by first selecting them, then holding down the middle mouse button over any of the selected nodes and dragging to the new position.

Class and Call Graph Display Controls

The Class or Call Graph main display can be manipulated with the controls found at the bottom of the Class or Call Graph window. Refer to Figure 7-30.

Zoom Option Menu and Zoom Buttons

The Zoom option menu and Zoom buttons let you choose the size of the class nodes in the Class or Call Graph display. You can choose one of the items from the option menu, or click on either of the two buttons to the right of the menu to zoom in or out on the graph.

Overview Button

Clicking on the Overview button opens the Context View window for the Class or Call Graph display. As you move the rectangular pane over a

condensed, schematic version of the graph displayed within the Class or Call Graph window, the display scrolls accordingly.

For example, Figure 7-32 shows the Call Context View window opened from a Call Graph. The area under the rectangular viewport is displayed in the Call Graph window. In a Class Graph Context View, the currently selected class node is highlighted in a different color from that of the other class nodes.

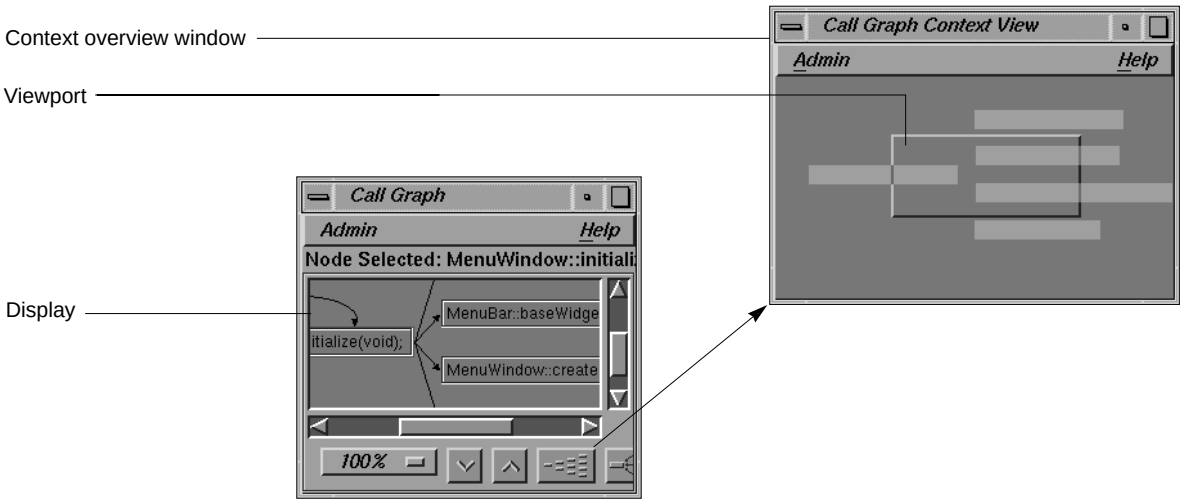


Figure 7-32 Call Graph Context Viewport of Call Graph Window

Multiple Arcs Button

Clicking on the Multiple Arcs button toggles the Class or Call Graph display between showing a single arrow for multiple instances of a relationship between two nodes, and showing a separate arrow for each instance.

Align Button

Clicking on the Align button tidies the nodes of the currently displayed graph, snapping them into an orderly configuration.

Rotate Button

Clicking on the Rotate button causes the graph displayed in the Class or Call Graph window to branch downwards from the top of the display, rather than from left to right, which is the default. Clicking on the button a second time returns the display to the default configuration.

Class Graph Window

The Class Graph window is invoked from the Views menu of the Class View window. It displays a graphical view of classes in the current fileset and their relationships. For details about the display, refer to “Class Graph and Call Graph Displays” on page 117. This section discusses the menu bar, keyboard accelerators, double-clicking, and the relationship option menu.

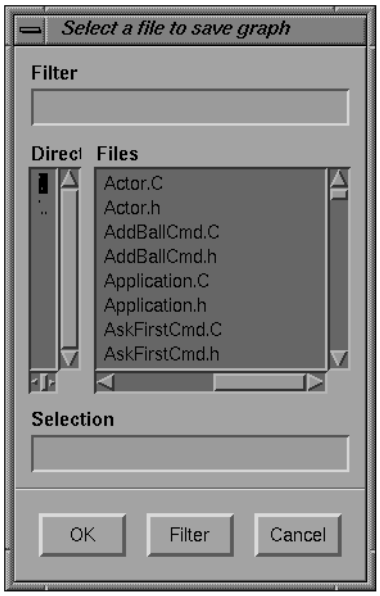


Figure 7-33 “Save Graph” Submenu of Admin Menu

Class Graph Menu Bar

The Class Graph window’s menu bar contains the three menus described in this section. By choosing the dashed line (the first item in each of the menus), you can “tear off” the menu from the menu bar, so that it is displayed in its own window, and is thus always visible.

Admin Menu

The Class Graph Admin menu contains two commands:

“Save Graph”	allows you to save the graph to a file. It brings up a file selection dialog. When you select your file and click on “OK,” it saves the graph as a PostScript® file with the name specified in “Selection.”(See Figure 7-33.)
“Close”	closes the Class Graph window when selected.

Views Menu

The Class Graph Views menu commands control which classes included in the current fileset are displayed in the Class Graph window. The choices are describe below.

"Show All" displays all classes included in the fileset as nodes, and their relations as arcs, as chosen from the relationship option menu (see Figure 7-30).

"Show All Related"
displays only those classes included in the chain of relations, which includes the current class.

"Show Butterfly"
displays only those classes that are the immediate relatives (for example, parents and children for an inheritance relation of the current class).

Help Menu

The Help menu contains commands that allow you to access online information about the C++ Browser. For more information on the help features of the C++ Browser, refer to "Help Menu" on page 103, and to Chapter 2, "Using On-line Help," in the *CASEVision Environment Guide*.

Double-Clicking

The current class that is displayed in the Class View from which the Class Graph window was launched is highlighted in the Class Graph window. Double-clicking with the left mouse on any class node in the Class Graph window makes that class the new current class in both the Class View and Class Graph windows.

Class Graph Relationship Option Menu

The relationship option menu lets you switch between viewing the different kinds of class relationships in a Class Graph window. The different graphs that you can display through the the option menu are listed below.

- "Inheritance"
- "Containment"
- "Interaction"
- "Friends"

Refer to “Views Menu” on page 99 for descriptions of these menu commands.

Keyboard Accelerators

To issue Class Graph commands directly from the keyboard, use the accelerators in Table 7-2. The accelerators are listed alphabetically by command.

Table 7-2 Keyboard Accelerators for Class Graph

Command	Ctrl + *key*
Show all	A
Show all related	R
Show butterfly	B

Call Graph Window

The Call Graph window is invoked from the Views menu of the Class View window or from the Queries on Methods pop-up menu for a selected method of the current class. It displays a graphical view of methods of the current class, as shown in Figure 7-34. For details about the display that are the same in the Class Graph window, see “Class Graph and Call Graph Displays” on page 117.

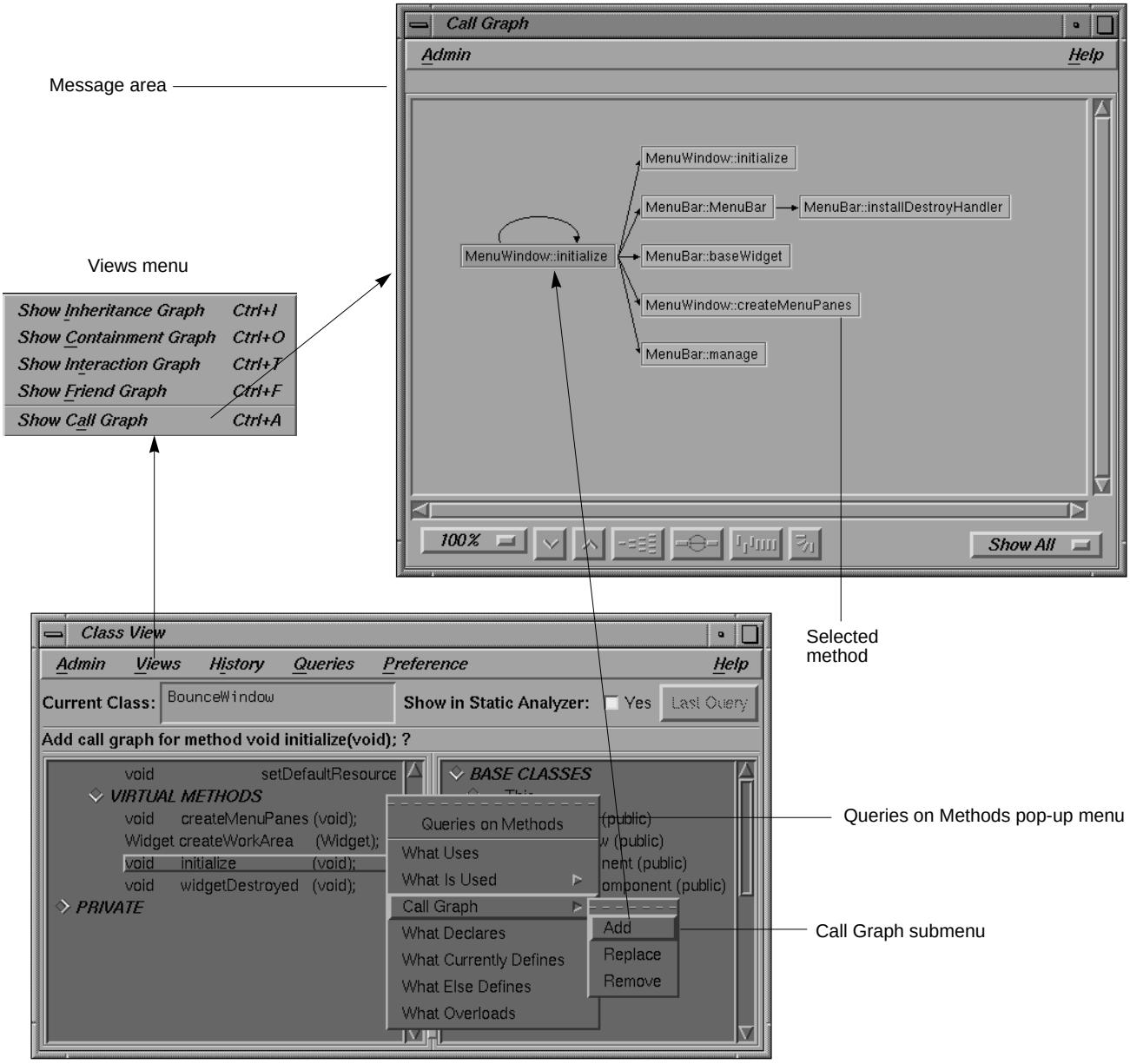


Figure 7-34 Working With the Call Graph Display

You add, replace, or remove methods by using the Queries on Methods pop-up menu in the member list of the Class View window. Refer to the Call Graph submenu description on page 110. Note that when you select a node in the display, a message area opens below the menu bar, showing the method with its argument list.

In a Call Graph window, double-clicking on any method node opens a Source View window on the code defining the method. The definition is highlighted in the source.

Call Graph Menu Bar

The Call Graph window's menu bar contains the two menus described in this section. By choosing the dashed line (the first item in each of the menus), you can "tear off" the menu from the menu bar, so that it is displayed in its own window, and is thus always visible.

Admin Menu

The Call Graph window's Admin menu contains the toggle and three commands described below.

"Show Arglist" toggle
: includes the argument list in the display of each method as shown in Figure 7-35. Clicking the toggle button turns it on. It is highlighted until clicked again, turning it off. Toggling "Show Arglist" shows all hidden nodes.

"Clear"
: removes all methods from the Call Graph window.

"Save Graph"
: allows you to save the graph to a file. It brings up a file selection dialog. When you select your file, it saves the graph to a postscript file. See Figure 7-33 for an example.

"Close"
: closes the Call Graph window.

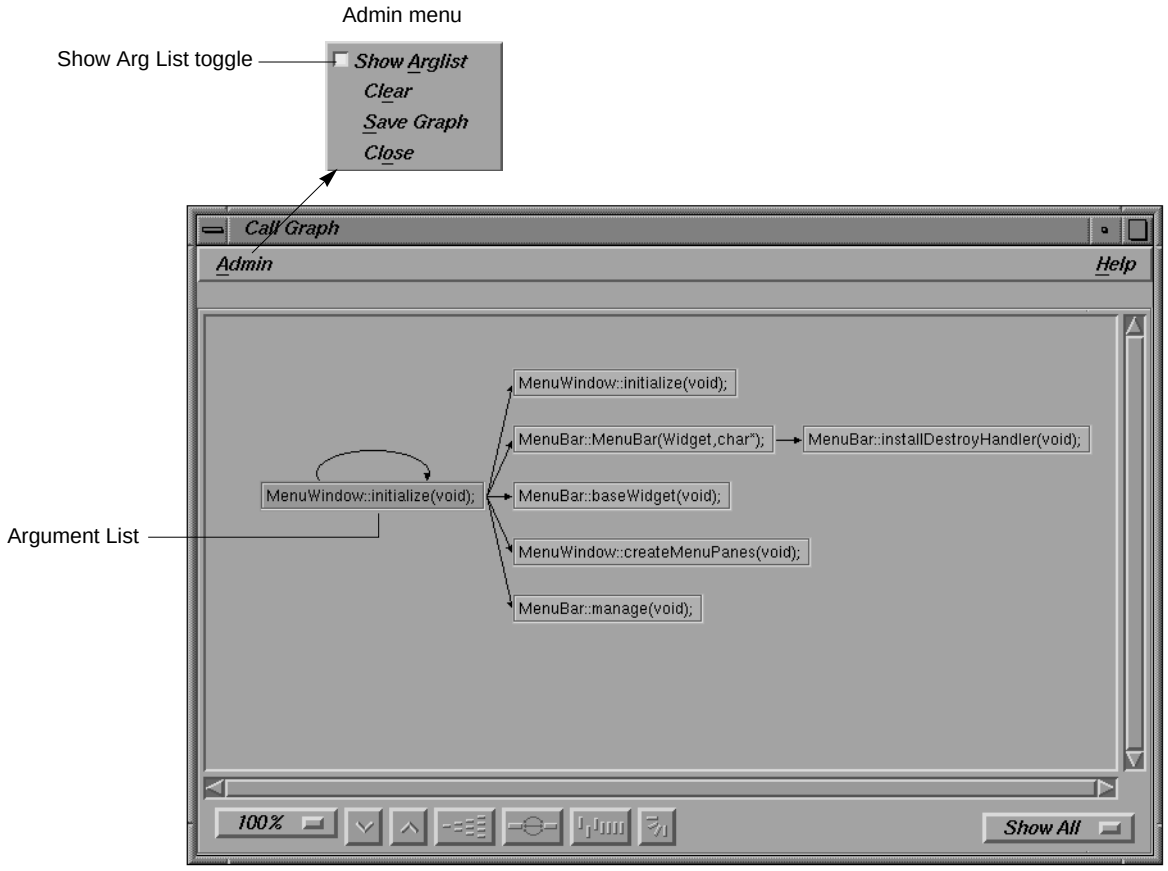


Figure 7-35 Call Graph With Show Arglist On

Help Menu

The Help menu contains commands that allow you to access online information about the C++ Browser. For more information on the help features of the C++ Browser, refer to “Help Menu” on page 103, and to Chapter 2, “Using On-line Help,” in the *CASEVision Environment Guide*.

Customizing the C++ Browser

The C++ Browser lets you customize your display and the way you work with reference pages (man pages). These formats are implemented as X application resources that you can redefine in your local *.Xdefaults* file. After editing it, run **`xrdb .Xdefaults`** and then reopen the Static Analyzer.

Customizing the Class View Lists

This section shows you how to customize the outline formats of Class View lists by applying your own keyword headers and rearranging the features of each list.

Member List Resource

The layout of the Class View member list is controlled by this resource:

```
Cvstatic*memberOrder
```

The general format of this resource is as follows:

Level-1-keyword: *HEADING* [*keyword*], *HEADING* [*keyword*],..; *Level-2-keyword*: *HEADING* [*keyword*], *HEADING* [*keyword*],...; *Level-3-keyword*: *HEADING* [*keyword*], *HEADING* [*keyword*],...;

The three level keywords are **Protection**, **Scope**, and **Member**. The order in which these are used determines the level of nesting in the outline list used for protection, scope, and member headings, respectively.

Headings may consist of any string you choose to describe the heading category. The headings listed with the level-1 keyword become top-level headings in the outline list, the level-2 headings appear indented under each of the level-1 headings, and the level-3 headings appear indented beneath each of the level-2 headings.

Each heading in a level has an associated keyword that determines the sort of items that appear under the heading. The allowable keywords are as follows for each associated level keyword:

```
Protection:   [public], [protected], [private]

Scope:        [instance], [static]

Member:       [type], [data], [method], [virtualmethod]
```

It is also possible to combine the types associated with two or more keywords under one heading by using the construction for any given heading:

HEADING [*keyword1*+*keyword2*+...]

You can also control whether a heading is expanded or collapsed when the browser starts up. Placing an asterisk (*) at the end of the heading string causes that heading to be collapsed by default:

*HEADING** [*keyword*]

The default assignment for the outline resource of the member list can be found in */usr/lib/X11/app-defaults/Cvstatic*. The contents of the file appear below:

```
Cvstatic*memberOrder: Protection: PUBLIC [public],
PROTECTED* [protected], PRIVATE* [private]; Scope: INSTANCE
[instance], STATIC [static]; Member: TYPE* [type], DATA
[data], METHODS [method], VIRTUAL_METHODS [virtualmethod];
```

Note: The sample above is a single line.

You can override this definition by placing your own definition in your local *.Xdefaults* file. For example, to make the display look like the sample in Figure 7-36, add this line:

```
Cvstatic*memberOrder: Member: IS (Type) [type], Data Members-
------------------- [data], Methods-------------------------
---- [method], Virtual Methods-------------------
[virtualmethod]; Scope: Non-Static [instance], Static
[static]; Protection: Private [private], Protected
[protected], Public [public];
```

Related Class List Resource

The layout of the Class View related class list is controlled by this resource:

```
Cvstatic*relationOrder
```

The construction of this resource is similar to that of the member list, but simpler:

HEADING [*keyword*], *HEADING* [*keyword*], . . .

The headings and keywords work as described for the member list, but there is no concept of level keywords in the related class list.

The allowable keywords for the related class list are as follows:

`[base]`, `[derived]`, `[uses]`, `[usedby]`, `[friendfunction]`, `[friend]`, `[friendof]`

Note: In the related class list, headings cannot contain multiple keywords, as they can in the member list. ♦

As in the member list, you can control whether a heading in the related class list is expanded or collapsed when the browser starts up. Placing an asterisk (*) at the end of the heading string causes that heading to be collapsed by default:

*HEADING** [*keyword*]

The default assignment for the related class list outline resource can be found in */usr/lib/X11/app-defaults/Cvstatic*, and is listed below for your convenience:

```
Cvstatic*relationOrder: BASE CLASSES [base], DERIVED CLASSES
[derived], USES [uses], USED BY [usedby], FRIEND FUNCTIONS
[friendfunction], FRIENDS [friend], FRIEND OF [friendof]
```

You can override this definition by placing your own definition in your local *.Xdefaults* file. For example, for the display shown in Figure 7-36, try this:

```
Cvstatic*relationOrder: Parent Classes [base], Child Classes
[derived], Used Classes [uses], User Classes [usedby],
Friend Functions [friendfunction], Friend Classes [friend],
Friend Of [friendof]
```

Other Class View List Resources

XWindows resources listed in this section, found in */usr/lib/X11/app-defaults/Cvstatic*, can be modified in your local *.Xdefaults* file. The default values are listed with each resource. You can set any *true* value to *false*.

`Cvstatic*completeClassName: true`
: enables ClassName completion; by typing a space in the current class field, you complete a class name from the list of classes in the fileset (if set to *true*, as it is by default).

`Cvstatic*showMessageArea: true`
: enables the message area in the Class View window (if set to *true*, as it is by default).

`Cvstatic*scream: true`
: enables warning beeps when there are 0 results for a query, or when a class name has more than one completion in the current class field (if set to *true*, as it is by default).

`Cvstatic*indentationWidth: 15`
: sets the indentation in the outline lists in pixels. Figure 7-36 shows the making the following change to the resource:

  `Cvstatic*indentationWidth: 10`

`Cvstatic*nameAlign: true`
: aligns names of the members under the same parent so that the type declarations and member (variable and function) names form left-justified columns (if set to *true*, as it is by default).

`Cvstatic*arglistAlign: true`
: aligns the argument lists of member functions under the same parent so they form a left-justified column (if set to *true*, as it is by default).

`Cvstatic*sort: true`
: sorts items in the outline lists based on the value of the entire string denoting an item (if set to *true*, as it is by default). For example, given two members, **`void f`** and **`int k`**, the C++ Browser lists **`int k`** before **`void f`** in the list.

`Cvstatic*nameSort: true`
: sorts items in the outline lists based on the string value of the name of a member (if set to *true*, as it is by default). For example, **`void f`** would be listed before **`int k`**).

Using both of the previous resources in conjunction sorts first by type and then by name, as shown in Table 7-3.

Table 7-3 Sort Resources for Outline Lists

sort	name	Sort effect
false	false	Members are in declaration order
false	true	Members are sorted based on the name and not on type or return type. This behavior is shown in Figure 7-36.
true	false	Members are sorted based on the their return type or type. Within the same return type, members appear in declaration order.
true	true	Members are sorted both on their type or return type and their name. This is the default behavior.

Figure 7-36 shows the Class View display using the sample resources set in *.Xdefaults*.

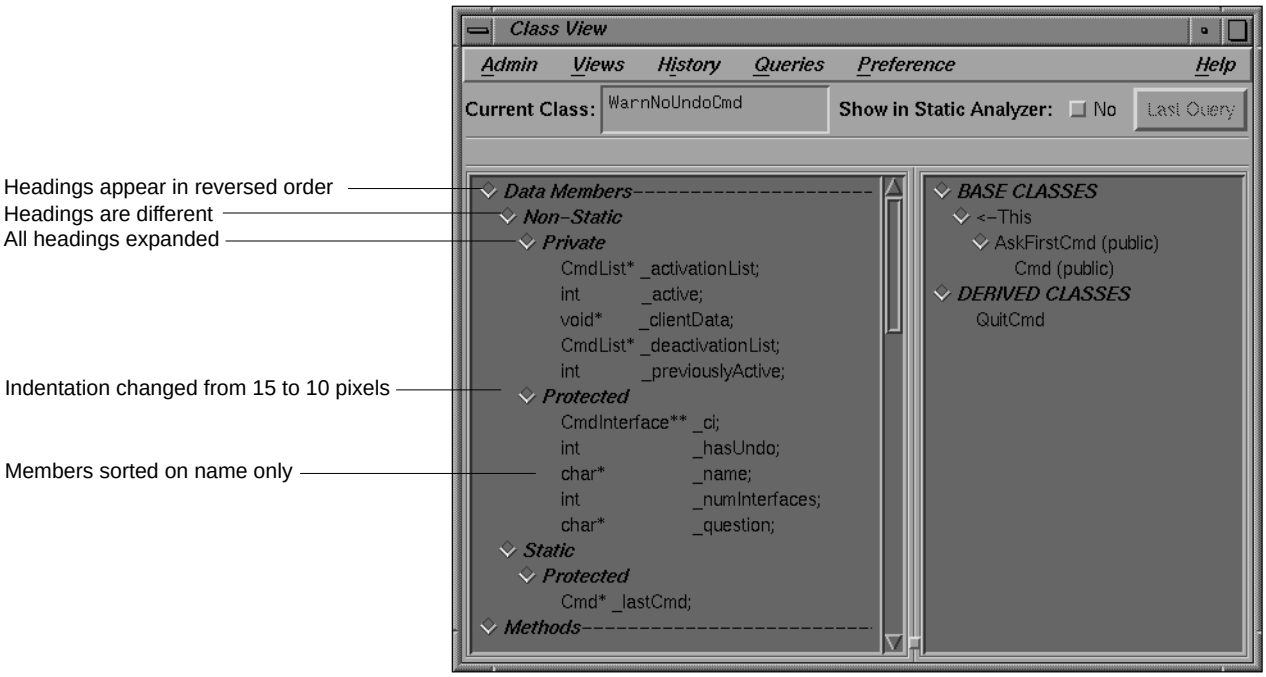


Figure 7-36 Customized Class View Display

Customizing Reference Page Generation

The resources in this section are associated with the Man Pages for Classes window, available from the Class View Admin menu item “Generate Man Pages.”

`Cvstatic*manPageDirPath: <default manpage directory path>`

The default is the current directory (.). To place generated reference pages in the *windTunnel* directory (that you have created) use this:

```
Cvstatic*manPageDirPath: ./manpage/windTunnel
```

`Cvstatic*manPageSuffix: .<suffix>`
: The default <suffix> is 3. The name of a reference page is <*class_name*>.3. To change the suffix to 4, use this:

```
Cvstatic*manPageSuffix: .4
```

`Cvstatic*manPageViewCommand: <commands>`
: Pressing the *View* button in the Man Pages for Classes window executes the command specified by this resource. The argument given is the set of reference pages for the classes that are selected. By default, *View* displays the most recently generated reference page in a read-only window. The default commands are:

```
Cvstatic*manPageViewCommand: winterm -H -c man -
d
```

`Cvstatic*manPageCopyRightMessage: <string>`
: The default string is “Copyright 1994 by Silicon Graphics.” A customized example is:

```
Cvstatic*manPageCopyRightMessage: Copyright
1994 by Fred Smythe
```

Glossary

Terms shown in *italics* indicate glossary items, in addition to document conventions such as buttons, file names, document titles.

<-This

C++ Browser current class as shown in the *Current Class* field. <-This appears in the related class list of the Class View window under the heading ***BASE CLASSES***.

access

Public, *Private*, or *Protected* members of a class.

base class

Parent of a class. In the C++ Browser, the parent of the *current class* is shown following <-*This* in the related class list.

beep

User interface warning. In the C++ Browser *Current Class* field, when attempting to complete a partial input string, the browser beeps if it finds more than one match.

calling structure

In the C++ Browser Call Graph, shows methods that call each other.

current class

Class you select to study in the C++ Browser. It is shown in the Class View window *Current Class* field. You can specify the class in the field or select it from the List of Classes, List of Classes Shown, or Class Graph windows.

current definition

C++ Browser term used with a method in the pop-up related class list query, “What Currently Defines.” The result of the query is the class that provides the current definition for the method.

data

In C++, data member of a class. A variable that contains state information for a class. A field of a class that is not a method.

database

Compiler-generated static analysis database built from a *fileset* in the Static Analyzer. Used by both the Static Analyzer and the C++ Browser. The C++ Browser accesses this database when it displays C++ class, *data*, and method information.

DCC

A native C++ compiler that allows you to use dynamic classes. See the *DCC(1)* reference page for more information.

derived class

In C++, child of the *current class*.

fileset

List of source file built by the Static Analyzer. Used to build a function and object *database* for the C++ Browser.

function

Refers to either a C function or a C++ member function.

instantiate

In C++, declare an object of type *classname*. The result is an instance or an object.

list of classes

C++ Browser chooser window that displays all the source files in the *fileset*. The selected class becomes the *current class* in the Class View and Class Graph windows.

list of classes shown

C++ Browser history window that lists each *current class* that you have selected, in chronological order.

member function

A C++ function that is a member of a class or structure data type. Also known as a method.

member list

In C++ Browser, lefthand outline list in the Class View. It contains members of the *current class*. They are sorted according to *access*, *scope*, and kind of member (*type*, *data*, *method*, or *virtual method*).

member

An object that contains either data or methods, or both, belonging to a class (class members).

method

C++ function that is a member of a class or structure data type. Also known as a member function.

multiple inheritance

Where a derived class inherits traits from more than one base class.

NCC

A native C++ compiler that uses the same compiler as DCC, but doesn't allow you to use dynamic classes.

object

C++ term. See *instantiate*.

outline button

In C++ Browser Class view, shows if entry is collapsed, contains query results, or is expanded.

private

In C++, a type of access to the class member that is restricted to the class in which it is defined, friend classes, or friend functions.

protected

In C++, a type of access to the class member that is restricted to the class (and all derived classes) in which it is defined, friend classes, and friend functions.

public

In C++, access is open to any method or function.

related class

A class (and friend functions) that is related to the *current class*.

scope

In the C++ Browser, a kind of member in the *member list*.

Smart Build

An option to the compiler where only those files that must be recompiled are recompiled.

type

In C++ Browser, data type that is a member of the *current class* displayed in the *member list*.

virtual method

In C++ Browser, member of the *current class*, displayed in the *member list*. Allows derived classes to provide different versions by redefining or overriding it.

Index

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

We'd Like to Hear From You

As a user of Silicon Graphics documentation, your comments are important to us. They help us to better understand your needs and to improve the quality of our documentation.

Any information that you provide will be useful. Here is a list of suggested topics to comment on:

- General impression of the document
- Omission of material that you expected to find
- Technical errors
- Relevance of the material to the job you had to do
- Quality of the printing and binding

Please include the title and part number of the document you are commenting on. The part number for this document is 007-2114-002.

Thank you!

Three Ways to Reach Us

The **postcard** opposite this page has space for your comments. Write your comments on the postage-paid card for your country, then detach and mail it. If your country is not listed, either use the international card and apply the necessary postage or use electronic mail or FAX for your reply.

If **electronic mail** is available to you, write your comments in an e-mail message and mail it to either of these addresses:

- If you are on the Internet, use this address: techpubs@sgi.com
- For UUCP mail, use this address through any backbone site: *[your_site]*!sgi!techpubs

You can forward your comments (or annotated copies of manual pages) to Technical Publications at this **FAX** number:

415 965-0964