ProDev™ WorkShop: Debugger
Reference Manual

Record of Revision

Version	Description
001	September 2002 Original publication. This book used to be included as part of the *ProDev WorkShop: Debugger User's Guide*. It is now a separate document.

Contents

Figures

Tables

About This Manual

This publication documents the ProDev WorkShop Debugger, released with the ProDev WorkShop tools running on IRIX systems.

The WorkShop Debugger is a source-level debugging tool that allows you to see program data, monitor program execution, and fix code for Ada (1.4.2 and older versions), C, C++, Fortran 77, and Fortran 90 programs.

This manual is part of a two-document set that describes the Debugger. This book serves as a reference guide to the Debugger features and the Debugger command line options. The other book in the set, the *ProDev WorkShop: Debugger User's Guide* contains user tutorials and other information about using the Debugger. All SGI documentation is available online at the following URL:

http://docs.sgi.com

This book contains the following chapters:

- Chapter 1, "Main View Window", page 1, describes the menus on the Main View Window.

- Chapter 2, "Additional Views in the Debugger", page 19, describes other views that are available from the Debugger, such as the **Execution View**, **Multiprocess View**, and **Source View**.

- Chapter 3, "Ada-specific Windows", page 29, describes the **Task View** and the **Exception View** windows.

- Chapter 4, "X/Motif Analyzer Windows", page 35, describes the X/Motif Analyzer windows.

- Chapter 5, "Managing Traps", page 55, describes the Trap Manager windows and their usage.

- Chapter 6, "Data Examination Windows", page 59, describes the windows used to examine the data used in your programs.

- Chapter 7, "Machine-level Debugging Windows", page 89, describes the windows used for machine-level debugging.

- Chapter 8, "Fix+Continue Windows", page 99, describes the windows used for the Fix+Continue utility.

- Chapter 9, "Debugger Command Line", page 109, describes the options used with the Debugger on the command line.

- Chapter 10, "Blocking Kernel System Calls", page 125, describes syscalls that block continued pthreads.

Related Publications

The following documents contain additional information that may be helpful:

- *C Language Reference Manual*

- *MIPSpro C++ Programmer's Guide*

- *MIPSpro C and C++ Pragmas*

- *ProDev WorkShop: Performance Analyzer User's Guide*

- *ProDev WorkShop: Debugger User's Guide*

- *ProDev WorkShop: Overview*

- *ProDev WorkShop: Static Analyzer User's Guide*

- *MIPSpro Fortran 77 Language Reference Manual*

- *MIPSpro Fortran 77 Programmer's Guide*

- *MIPSpro Fortran Language Reference Manual, Volume 1*

- *MIPSpro Fortran Language Reference Manual, Volume 2*

- *MIPSpro Fortran Language Reference Manual, Volume 3*

- *MIPSpro Fortran 90 Commands and Directives Reference Manual*

- *dbx User's Guide*

Obtaining Publications

You can obtain SGI documentation in the following ways:

- See the SGI Technical Publications Library at: `http://docs.sgi.com`. Various formats are available. This library contains the most recent and most

comprehensive set of online books, release notes, man pages, and other information.

- If it is installed on your SGI system, you can use InfoSearch, an online tool that provides a more limited set of online books, release notes, and man pages. With an IRIX system, select **Help** from the Toolchest, and then select **InfoSearch**. Or you can type infosearch on a command line.

- You can also view release notes by typing either grelnotes or relnotes on a command line.

- You can also view man pages by typing man *title* on a command line.

Conventions

The following conventions are used throughout this document:

Convention	Meaning
command	This fixed-space font denotes literal items such as commands, files, routines, path names, signals, messages, and programming language structures.
manpage(*x*)	Man page section identifiers appear in parentheses after man page names.
variable	Italic typeface denotes variable entries and words or concepts being defined.
user input	This bold, fixed-space font denotes literal items that the user enters in interactive sessions. (Output is shown in nonbold, fixed-space font.)
[ ]	Brackets enclose optional portions of a command or directive line.
...	Ellipses indicate that a preceding element can be repeated.

GUI   This font denotes the names of graphical user interface (GUI) elements such as windows, screens, dialog boxes, menus, toolbars, icons, buttons, boxes, fields, and lists.

Reader Comments

If you have comments about the technical accuracy, content, or organization of this document, contact SGI. Be sure to include the title and document number of the manual with your comments. (Online, the document number is located in the front matter of the manual. In printed manuals, the document number is located at the bottom of each page.)

You can contact SGI in any of the following ways:

* Send e-mail to the following address:

  techpubs@sgi.com

* Use the Feedback option on the Technical Publications Library Web page:

  `http://docs.sgi.com`

* Contact your customer service representative and ask that an incident be filed in the SGI incident tracking system.

* Send mail to the following address:

  Technical Publications
  SGI
  1600 Amphitheatre Parkway, M/S 535
  Mountain View, California 94043–1351

* Send a fax to the attention of "Technical Publications" at +1 650 932 0801.

SGI values your comments and will respond to them promptly.

Main View Window

This chapter provides an overview of the main menus shown on the Main View window.

Primary Components of the Main View Window

The Main View window is shown in Figure 1-1.

Figure 1-1 The Main View Window

The Main View window contains a menu bar, from which you can perform a number of actions and launch windows. The screen and menu bar contains the following items, which are discussed in detail in this chapter:

- "Admin Menu", page 6
- "Views Menu", page 8

- "Query Menu", page 9

- "Source Menu", page 10

- "Display Menu", page 11

- "Perf Menu", page 12

- "Traps Menu", page 13

- "Fix+Continue Menu", page 14

- "Help Menu", page 17

When you start the Debugger with an executable file, the Main View window displays, loaded with source code, ready to execute your program with your specified arguments. Most of your debugging work takes place in the Main View window, which includes the following main areas:

- A **menu bar** for performing debugger functions.

- A **control panel** for specifying and controlling program execution.

- A **source code display area** which displays the code for the program you are debugging.

- A **source filename** field which tells gives you the path to the file displayed in the source code display area.

- A **status area** for viewing the current status of the program.

- The Debugger **command line** in which to enter debugging commands.

The Main View window contains the following items:

- **Command** text field: Displays full pathname of the executable file that you are currently debugging, including any run-time arguments.

- **Debug** button: Allows you to toggle among various modes. Right-click on the **Debug** button to display the **State Indicator menu** where the following mode choices are available:

  – **Debug** runs the Debugger in Debug mode with no performance tools enabled.

  – **Performance** mode causes performance data to be gathered and instrumented code to be generated for performance analysis while using the Debugger.

- **Purify** mode activates the Purify memory corruption analysis tool. The code displayed in the Main View window, **Source View** window, and so forth is code generated by **Purify**. (This option appears only if Purify is installed on your system. Purify is not an SGI product nor is it part of the WorkShop package. It is a product of Rational Software and is neither available from nor supported by SGI.)

- The lock (**Stay Focused/Follow Interesting**) button: If the lock icon is locked, it indicates that the focus of Main View will attempt to stay focused on this thread. If the lock is unlocked, the debugger follows the interesting thread.

- **All/Single** button: If set to **All**, the **Cont**, **Stop**, **Step**, **Next**, and **Return** actions apply to all processors or threads. If set to **Single**, then only the currently focused process or thread will be acted on.

- **Cont** button: Continues execution of the current process or all processes in the program. When you click on the **Cont** button, the program runs either to a breakpoint exception or to termination. This button is active only after the running process(es) has stopped. If the program has not been run or has been killed, the **Cont** button is grayed out. If the target program has not yet started executing, use the **Run** button to start execution.

- **Stop** button: Stops execution of the current running process(es). This button is valid only when a process is running; otherwise the button is grayed out. Traps can also be set to stop the program at a specific location or on a particiular condition.

- **Step** button: Executes the code involving a single source line of the current process. If a function is encountered in the source line, or is the source line in a subroutine call, the process steps into a function or subroutine call and stops at the first executable line in the funciton or subroutine. The **Next** button can be used to continue to the next source line in the current file. If a trap is encountered while executing the step into command, the process is stopped where the trap was fire. The **Step** button is active only after the running process has stopped; otherwise the button is grayed out.

  When you right-click on the Step button, a menu pops up to allow you to choose the number of source lines to be stepped. If you choose the N menu entry, a dialog window is opened to allow you to enter a step value.

- **Next** button: Executes the code involving a single source line of the current process. The current process continues to the next source line in the current file, and does not count any statements in functions that may be encountered in the

source file. If the source line is a subroutine call, the process stops at the next source line in the current file. The **Step** button can be used to step into a function or subroutine call and stop at the first executable line in the function or subroutine. If a trap is encounrtered while executing the step over command, the process is stopped where the trap was fired. The **Next** button is active only after the running process(es) has stopped; otherwise the button is grayed out.

When you right-click on the **Next** button, a menu pops up to allow you to choose the number of source lines to be stepped. If you choose the **N** menu entry, a dialog window is opened to allow you to enter a step value.

- **Return** button: Continues execution of the process until the current function being executed returns. The process is stopped immediately upon returning to the calling function. All code within the current function is executed as usual. If a breakpoint is encountered, the action is canceled and the process is stopped where the breakpoint was fired. You can use this button only after the running process(es) has stopped; otherwise, the button is grayed out. This action is not allowed if the executable is instrumented for performance analysis.

Note: For IRIX 6.5, this command button always returns all pthreads.

- **Sample** button: Allows you to collect process state data to be used by the Performance Analyzer for program evaluation. You can use this button only when the process(es) is running and the **Enable Data Collection** mode is set on the **Performance** panel; otherwise, the button is grayed out. See the *ProDev WorkShop: Performance Analyzer User's Guide* for details.

- **Print** button: Prints, in the **Source View** window, the value of any highlighted text, or the source pane of the Main View window.

- **Kill** button: Kills the currently running process or all running processes in your debug session by sending the process(es) the equivalent of a `kill -9` signal. You can use this button if the process(es) is running or stopped; otherwise the button is grayed out.

- **Run** button: Runs the program that you are currently debugging or all programs. After the initial run, allows you to rerun the program(s) while maintaining any breakpoints and command line arguments you have set.

- **Status** area: Displays information about the process that you are debugging, such as process id, thread id, function name, list of arguments, location of the PC, and so forth,

- Source Code area: Displays the source code for the program that you are currently debugging.

- Annotation column: Clicking in this area displays information specific to a line number, such as breakpoints, location of the PC, and so forth.

- **File** text field: Displays the name of the file shown in the source code area.

  The down arrow indicates that a pulldown menu is available. The menu has a list of files that have recently been accessed and displayed. Clicking on a menu item displays the corresponding file in the Source View display area.

- Command line area: Area of the Main View window where you can enter Debugger commands and view line-mode debugger message, and in which Debugger messages are displayed.

- Show/Hide annotations button: This button is visible only when you run or load a performance experiment (see the *ProDev WorkShop: Performance Analyzer User's Guide* for more information on the performance tools). This is a toggle button that shows or hides performance related annotations.

Admin Menu

The **Admin** menu in the Main View window performs administrative and general management tasks dealing with processes, windows, and user preferences. The **Admin** menu provides the following selections:

- **Library Search Path**: Controls where the Debugger looks for DSOs when you invoke the Debugger on an executable or core file. The **Library Search Path** dialog allows you to reset the LD_LIBRARY_PATH and _RLD_ROOT environment variables. You can also reset _RLD_LIST to control the set of DSOs that are used by the program. See the rld(1) man page for more information on these variables. Any changes you make to these variables are propagated into **Execution View** when you run the program.

  There are two ways to open this dialog. First, select **Admin > Library Search Path** from the Main View window menu bar, or the **Library Search Path** dialog opens automatically if you invoke the Debugger on an executable or core file and it is unable to find all of the required DSOs. In this case, an annotated list of required DSOs displays at the top of the dialog box with such status messages as OK, Error: Cannot find library, or Error: Core file and library mismatch (which indicates that the Debugger found a DSO that did not match

the core file). Below this list are three to nine fields in which you can modify the value of the corresponding named environment variable.

- **Remap Path**: Opens a dialog that allows you to enter a new pathname.

- **Multiprocess Explorer**: Displays the **Multiprocess Explorer** window, which allows you to control processes and threads. You should note that if you exit from **Multiprocess Explorer**, you exit from your debugging session. For additional information about **Multiprocess Explorer**, refer to "Multiprocess Explorer", page 19.

- **GLdebug**: Provides a toggle to turn on GLdebug. GLdebug is a graphical software tool for debugging application programs that use the IRIS Graphics Library (GL). GLdebug locates programming errors in executables when GL calls are used incorrectly.

- **Attach/Switch Process**: Changes the current process or attaches to a process. You are queried for the new process ID. You can select one from the list of items presented, type one in or paste one in from another window. Switching processes changes the session.

- **Load/Switch to Executable**: Changes the current executable or loads an executable. This option also allows you to debug a different core file.

- **Detach**: Releases the process from the Debugger. This allows you to make changes to the source code. You must detach the process before you recompile the program.

- **Load Settings**: Allows you to use the previously saved preference settings from a file you choose in the **Load Settings** dialog.

- **Save Settings**: Allows you to save the current preference settings to an initialization file used when the Debugger is first started, or any file you choose through this dialog. These can include such items as window sizes, current views, window configurations, and so on.

- **Iconify**Iconifies all session views.

- **Raise**: Brings all session view windows to the foreground and displays any iconified windows.

- **Launch Tool**: Allows you to run other WorkShop tools. You can switch to the other tools by selecting **Build Analyzer**, **Static Analyzer**, **Performance Analyzer**, or **Tester**. Selecting **Debugger** allows you to start another debugging session. If you have ProDev ProMP (formerly called WorkShop Pro MPF) installed on your system, the **Parallel Analyzer** selection is also available.

- **Close**: Closes the Main View window.

- **Exit**: Closes all views in the session and terminates the session.

Views Menu

The **Views** menu in Main View window provides the following selections for viewing the process(es) and their corresponding data:

- **Array Browser**: Displays values from an array or array-slice in a two-dimensional spreadsheet and optionally in a three-dimensional representation; that is, a bar graph, surface, multiple lines, or points in space. These help you pick out bad data more readily. Arrays can contain up to 100 x 100 elements.

- **Call Stack**: Displays the call stack along with parameters to the calls. If you double-click an entry in the stack, you switch the current context to that entry and you can check the state of variables.

- **Data Explorer**: Displays data structures in a graphical format. You can de-reference pointers by double-clicking.

- **Data View**: brings up a Data Window.

- **Disassembly View**: Displays assembly code corresponding to the source code.

- **Exception View**: Displays an Ada-specific window used for exception handling.

- **Execution View**: Displays the iconified **Execution View** window, which handles the input and output of the target process.

- **Expression View**: Evaluates expressions in Fortran, C, or C++. To enter an expression, select it in the source code display and paste it into the **Expression View** field, using the middle mouse button.

- **File Browser**: Displays a list of source files and library routines used by the current executable. Double-click a source file in the list to load it directly into the source display area in Main View or **Source View** windows. The **Search field** allows you to find files in the list quickly.

- **Memory View**: Displays the value at a given memory address.

- **Multiprocess Explorer**: Displays a window which allows you to control processes and threads.

- **Process Meter**: Monitors the resource usage of a running process without saving the data. (Used with the Performance Analyzer.)

- **Register View**: Displays the values stored in the hardware registers for the target process.

- **Signal Panel**: Displays the signals that can occur. You can specify which signals trigger traps and which are to be ignored.

- **Source View**: Displays source code. Allows you to set traps, perform searches, and inspect source code without losing information in the Main View window.

- **Syscall Panel**: Allows you to set traps at the entry to or exit from system calls.

- **Task View**: Brings up an Ada-specific view that provides task and callstack information for processes.

- **Trap Manager**: Allows you to set, edit, and manage traps. The Trap Manager is used by both the Debugger and the Performance Analyzer.

- **Variable Browser**: Displays values of local variables and parameters for the current context.

- **X/Motif Analyzer**: Provides you with specific debugging support for X/Motif applications. There are various examiners for different X/Motif objects, such as widgets and X graphics contexts, that might be difficult or impossible to inspect using ordinary Debugger functionality.

Query Menu

The **Query** menu allows you to perform some of the queries available in the Static Analyzer. These queries are convenient if you have previously built a `cvstatic` fileset. However, if you need to build the fileset from scratch, the process becomes more involved. For complete information about using the Static Analyzer, see the *ProDev WorkShop: Static Analyzer User's Guide* and the `cvstatic`(1) man page.

With a current fileset, you can double-click any defined entity in the source code, select the **Where Defined?** option from the submenu appropriate to its type, and the source code display area scrolls to the location where the item is defined.

Source Menu

The **Source** menu in the Main View window provides the following selections to manage source code files:

- **Open**: Loads a source file.

- **Open Recent**: Provides you with a popup dialog that gives you a selection of recently-opened files from which to choose.

- **Save**: Records changes made during the debugging session to the source file. You must first select **Make Editable**, which appears in the **Source** menu when the file is read-only.

- **Save As**: Records changes made during the debugging session to the source file under a different filename. You must first select **Make Editable**, which appears in the **Source** menu when the file is read-only.

- **Save As Text**: Records the information in the display area as a text file.

- **Insert Source**: Inserts the text of a file within your current file. You must first select **Make Editable**, which appears in the **Source** menu when the file is read-only. You must first select **Make Editable**, which appears in the **Source** menu when the file is read-only.

- **Fork Editor**: Starts your default editor on the current file. The default editor is determined by the editorCommand resource in the app-defaults file. The value of this resource defaults to wsh -c vi +%d, which means run vi in a wsh window and scroll to the current line. If the editor allows you to specify a starting line, enter %d in the resource to indicate the new line number.

- **Recompile**: Displays the **Build View** window, which allows you to compile the source code associated with the current executable.

- **Make Read Only / Make Editable**: Toggles the source code displayed between read-only and writable states so that you can edit your code.

- **Search**: Searches for a literal case-sensitive, literal case-insensitive, or regular expression. After you have set your target and clicked **Apply** (or pressed Enter), each instance is marked by a search target indicator in the scroll bar. You can search forward or backward in the file by clicking the **Next** and **Prev** buttons. You can also click an indicator with the middle mouse button to scroll to that point. Clicking **Reset** removes the search target indicators.

- **Go To Line**: Allows you to scroll to a position in the source code by specifying a line number. **Go To Line** brings up a dialog box.

  You can enter a line number or use the slider at the top of the box to select a line number. You do not have to display line numbers to use this feature.

- **Versioning**: Provides access to the configuration management tool, if you have designated one.

  Type the following at the **Execution View** prompt:

  /usr/sbin/cvconfig [rcs | sccs | ptools | clearcase]

  > **Note:** You must have `root` permissions to run `cvconfig`.

  The **Versioning** submenu appears.

  Selecting any submenu option displays a shell in which you can access the configuration management tool. The following selections are available on the submenu:

  – **CheckIn** — Saves the source file and checks it into the database as a new version.

  – **CheckOut** — Recalls the source file from the tool's database if you have the proper authority, locks it, and makes it editable.

  – **UncheckOut** — Cancels the checkout, with no changes registered.

Display Menu

The **Display** menu in the Main View window provides the following selections to annotate the displayed source code:

- **Show Line Numbers**/**Hide Line Numbers**: Displays or hides line numbers in the annotation column corresponding to the source code.

- **Show Toolbar**: Allows you to choose the format type for the toolbar. The options are **Text Only**, **Icons Only**, or **Icons and Text**. **Text Only** was the only format for the toolbar prior to release 2.9. **Icons Only** and **Icons and Text** are options which display icon options for more visual debugging.

- **Show Tooltips**/**Hide Tooltips**: This menu item enables or disables the context sensitive pop-up help option. Some of the key menu items, buttons, and data entry areas have pop-up help statements attached to them to give the user hints on what to use them for or how to use them.

- **Preferences**: Displays the **Annotations Preferences** dialog box, which allows you to show or hide column annotations and menus specific to the different WorkShop tools. If you have purchased ProDev ProMP, you can display and manipulate loop indicators. The **Performance** toggle displays experiment statistics. The Tester module allows you to see coverage statistics. Turning off the **Performance** toggle deletes the performance annotations from the **Source View**.

- **Hide Icons**/**Show Icons**: Hides or displays the annotation column, which is located to the left of the source code display area.

- **Expression Evaluation**: Toggles between expression evaluation or auto-evaluation of expressions

- **Show Toolbar**: Toggles between showing the toolbar as icons, text, or a combination of both.

Perf Menu

The **Perf** (Performance) menu includes the following menu selections:

- **Select Task** submenu: Allows you to choose the task for your performance analysis. You may select only one task per performance analysis run. If none of the given tasks satisfy your requirements, you can choose **Custom**, which brings up the configuration dialog open to the **General** tab. From here, you can design your own task requirements.

- **Examine Results**: Launches the Performance Analyzer. For complete information about the Performance Analyzer, see the *ProDev WorkShop: Performance Analyzer User's Guide*.

- **Configs**: Brings up the configuration dialog open to the **Runtime** tab. The dialog opens with the **Experiment Directory** text field filled in with a default value. The Performance Analyzer provides a default directory named test0000. If you use the default or any other name that ends in four digits, the four digits are used as a counter and are incremented automatically for each subsequent experiment.

Traps Menu

The **Traps** menu offers the **Set Trap** and **Clear Trap** submenus and the **Group Trap Default** and **Stop All Default** menu options.

The **Set Trap** submenu offers menus for managing breakpoints and sample points. The following submenu selections are available:

- **Stop**: Sets a breakpoint at a designated line in your source code. To set a breakpoint at a line displayed in the Main View or **Source View** windows:

  1. position the cursor on the appropriate line in the source code display area

  2. select the **Set Trap** submenu

  3. choose the **Stop** option

  The preferred method for setting a breakpoint is to click in the annotations area of the Main View window, across from the line at which you want to set the breakpoint.

- **Stop At Function Exit**: Sets a breakpoint at the end of a function. To set a breakpoint at a function exit, click on the function name in the source code display area and select the **Set Trap** submenu, then choose the **Stop At Function Exit** option.

- **Sample**: Sets a sample trap at a line displayed in the Main View or **Source View** windows. To set a sample trap:

  1. highlight on the appropriate line

  2. pull down the **Set Trap** submenu

  3. select the **Sample** option

- **Sample At Function Entry**: Sets a sample trap at the beginning of a function. To set the sample trap, highlight the function name in the source code display area, then pull down the **Set Trap** submenu and select the **Sample At Function Entry** option.

- **Sample At Function Exit**: Sets a sample trap at the end of a function. To set the sample trap, highlight the function name in the source code display area, then pull down the **Set Trap** submenu and select the **Sample At Function Exit** option.

The **Clear Trap** submenu contains selections that allow you to delete a trap on the line containing the cursor. You must designate **Stop** or **Sample** trap type, since both

types can exist at the same location, appearing superimposed on each other. The following submenu selections are available:

- **Stop**: Designates the stop trap type.

- **Sample**: Designates the sample trap type.

The last two menu options allow you to specify the following items:

- **Group Trap Default**: Interacts with **Source View**. If set to true, all subsequent **Source View** trap requests are group traps. That is, all members of the process group apply this trap. This option is the same as typing `stop pgrp in` *filename* from the command line. Default is false unless you are using IRIX 6.5 pthreads, when the implied setting is always true.

- **Stop All Default**: Interacts with **Source View**. If set to true, all subsequent **Source View** trap requests apply the `Stop All` command to the trap. That is, whenever this trap is encountered, all other members of the process group also are stopped. This option is the same as typing `stop all in` *filename* from the command line. Default is false unless you are using IRIX 6.5 pthreads, when the implied setting is always true.

If both of the default options are set to true, it is the same as typing `stop all pgrp in` *filename* from the command line.

Fix+Continue Menu

The **Fix+Continue** menu offers the following menu selections:

- **Edit**: Allows you to edit text using the Debugger editor.

- **External Edit**: Allows you to edit text by using an external editor. The default editor is `vi`, but can be changed by using the **Set Edit Tool** pop-up menu in the **Admin** menu of the **Status** window. See "Fix+Continue Status Window", page 99, for further information.

- **Parse and Load**: Compiles your modified program and loads it for execution. You can execute the modified program by clicking on the **Run** or **Continue** buttons in the Main View window.

- **Show Difference** submenu: Allows you to see the difference between the original code and your modifications. The submenu options are:

- **For Function**: Opens a window that shows you the differences between the original 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 **Fix+Continue Preferences Dialog** that allows you to set the tool that displays code differences. The default is xdiff(1).

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

- **Save As**: Allows you to save your changes to a file. You can save changes to the current source file (the default) or to a separate file.

- **Save All Files**: Launches the **Save File+Fixes As** dialog that allows you to update the current session and save all the modifications.

- **View** submenu: Allows you to change to different views. Fix+Continue supports status, message, and build environment windows. This submenu contains the following options:

  - **Status Window**: Launches the **Fix+Continue Status** window. See "Fix+Continue Status Window", page 99, for more information.

  - **Message Window**: Launches the **Fix+Continue Message** window. See "Fix+Continue Message Window", page 102, for more information.

  - **Build Environment Window**: Launches the **Fix+Continue Build Environment** window. See "Fix+Continue Build Environment Window", page 103, for more information.

- **Preferences** submenu: Allows you to set your Fix+Continue preferences. It contains the following options:

  - **Show Preferences**: Launches the **Fix+Continue Preference Dialog** that displays preferences currently enabled for the session, and allows you to change the settings. The following preferences are available through the dialog:

    - **External Editor Command** text field that allows you to choose your text editor. The default is vi.

    - **File Difference Tool** text field allows you to choose the tool to use when comparing code. The default is xdiff(blank).

- **Copy Traps On Previous Definition** toggle allows you to edit and parse code. When Fix+Continue copies traps from the old definition to the new one by mapping old lines to new lines. (This mapping is the same as what can be generated using the UNIX `diff` utility.) If **Copy Traps On Previous Definition** is on and the mapped line the new definition is modified, then Fix+Continue will look at the switch.

- **Copy Traps Even On Changed Lines** toggle causes the debugger to copy traps onto a mapped line.

- **Continue Even If Line Has Changed** toggle allows you to edit and compile code in which your program is currently stopped. Fix+Continue can continue in the new definition provided some conditions are satisfied. The line from which the program continues depending on the mapping from the line in which it stopped. In case it can continue in the new definition from a line which you have modified, Fix+Continue consults this toggle to determine whether to continue in the new or old definition. This toggle allows you to override the default behavior.

- **Warn Unfinished Edits Before Run** toggle pops up a warning dialog before a run if you have unfinished edits.

- **Warn Unfinished Edits Before Continue** toggle pops up a warning dialog before a continue if you have unfinished edits.

- **Save deactivated code during File Save** toggle save old code. The Fix+Continue file save substitutes new definitions in place of old ones. If you want to save your original functions in the same file, this switch allows you to save the old (original or compiled) code under an `#ifdef`. When you compile, the old code does not get compiled. You can manually edit the source to use the old definition in any way you desire.

  – **Reset To Factory Defaults**: Sets preferences to the installed defaults.

  – **Save Preferences**: Brings up the **File** dialog that allows you to save your preferences to a file.

  – **Load Preferences**: Brings up the **File** dialog that allows you to load preferences from a file.

- **Cancel Edit**: Takes you out of edit mode and cancels any changes you have made.

- **Delete Edits**: Deletes any modifications that you made.

Keyboard Accelerators

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

Table 1-1 Fix+Continue Keyboard Accelerators

Command	Key Sequence
Cancel Edit	Alt+Ctrl+q
Edit	Alt+Ctrl+e
Parse And Load	Alt+Ctrl+x

Help Menu

The **Help** menu provides the following options:

- **Click for Help**: Provides information on the selected window or menu.

- **Overview**: Provides general information on the current tool.

- **Index**: Displays the entire list of help topics, alphabetically, hierarchically, or graphically.

- **Keys & Shortcuts**: Lists the keys and shortcuts for the current tool.

- **Product Information**: Provides copyright and version number information on the tool.

PC Menu

The **PC** (program counter) menu in previous releases was location on the Main View menu. As of the WorkShop 2.9.2 release, the PC option is now a dynamic menu and can be accessed by running an executable file and holding down the right mouse button on a blank area in the source file. When you do so, the following options appear:

- **Continue To**: Continues the process to the selected point in the program unless some other event interrupts. Select a line by clicking on it. The process must be stopped before you can use **Continue To**.

- **Jump To**: Goes directly to a selected point within the same function, jumping over intervening code. Then the Debugger waits for a command to resume execution. Select a line by clicking on it.

Additional Views in the Debugger

This chapter discusses some of the additional views available through the Debugger: the **Execution View**, **Multiprocess Explorer**, **Source View**, and **Process Meter**.

Execution View

The **Execution View** window is a simple shell that allows you to set environment variables and inspect error messages. If your program is designed to be interactive using standard I/O, this interaction takes place in the **Execution View** window. Any standard I/O that is not redirected by your Target command is displayed in the **Execution View** window. **Execution View** is launched (and iconified) automatically with the Debugger.

Multiprocess Explorer

WorkShop supports debugging of multiprocess applications, including pthreaded programs and processes spawned with either `fork` or `sproc` commands.

Multiprocess debugging is supported primarily through the **Multiprocess Explorer** window. To display this window, select **Multiprocess Explorer** from the **Admin** menu of the Main View window. **Multiprocess Explorer** window displays a hierarchical view of your pthreaded application. Pthreaded processes are marked with a folder icon. Clicking the folder changes the view to show that process's pthreads. Clicking on a thread opens a call stack for that thread.

For each process or thread, clicking the right mouse button brings up a menu that applies to the selected item. This menu is a duplicate of the **Process** menu. See "Controlling Multiple Processes", page 25 for more information.

Status of Processes

When the **Multiprocess Explorer** window comes up, it defaults to a list of the status of all processes in the process group.
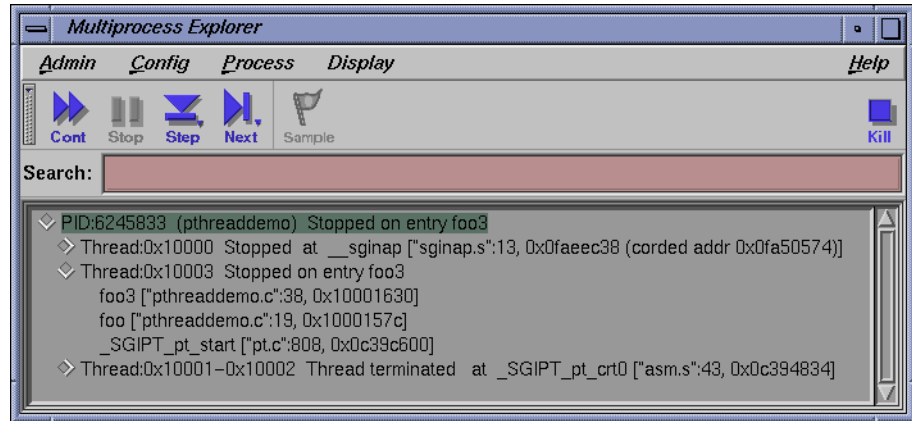
Figure 2-1 Multiprocess Explorer: reference by process

This view includes the following information:

- **OmpThread**: OmpThread number or Rank. The MPI rank number is shown if this is an OpenMP or MPI application.

- **PID**: Shows the process identifier (PID).

- *executable name*: Name of the executable for this process.

- **Process Status** for the process. This typically matches the process status line of the Main View.

- **Function/PC**: Indicates the current function and program counter (PC) for any stopped processes. This typically matches the process status line of Main View.

If pthreads are present in the process, clicking the **open** widget to the left of the **Process** line displays any pthreads that the process may have and the following information:

- **thread**: The thread number for each thread.

- **State**: Shows pthread state depending on how the preferences are set on the **Multiprocess Explorer** menu, brought up by selecting **Preferences** from the **Config** menu and using the **Hide Thread State** toggle. **State** is the kernel-level status and is not displayed by default.

- **Status**: status is user-level and is always displayed.

The Multiprocess Explorer also allows viewing by process/thread status. This groups processes and/or threads at the same status and location under one status node and reduces the amount of information displayed on the screen when there are very large numbers of processes and/or threads. This is most useful for large OpenMP and MPI jobs.
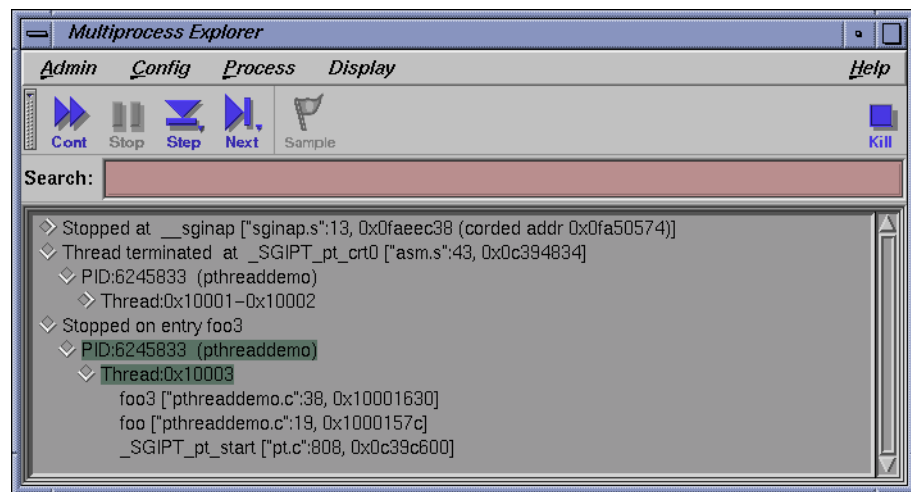

Figure 2-2 Multiprocess Explorer: reference by status

This view includes the following information:

- **top level status node**: This typically matches the process status line of Main View. Selecting this node's **open** widget opens the Status node to reveal a list of process' threads that are at this particular status.

- **process nodes**: the process nodes are directly under the Status nodes and are displayed as **PID:***number* followed by the executable name for the process.

If a process contains pthreads, toggling the process node widget to the left of the process number displays the pthreads as **thread:***number* and the pthread state if the **Hide Thread State** preference is not selected in the **Multiprocess Explorer** preferences.

The following **Status** and **State** conditions are possible:

Status	State
Running	RUNNING
Stopped	RUNNING
Stopped on breakpoint	RUNNING (but at a trap pc)
Waiting to terminate	JOIN
Thread terminated	DEAD
Waiting on kernel	READY
Waiting on mutex	MUTEX-WAIT
Sleeping in system call	RUNNING

Multiprocess Explorer Control Buttons

The **Multiprocess Explorer** window uses the same control buttons as are in the Main View window with following exceptions:

- There are no Run, Return, or Print buttons in the **Multiprocess Explorer**.

- The buttons in this view apply to all processes as a group.

- Using a control button in the **Multiprocess Explorer** window has the same effect as clicking the button in each process's Main View window.

These buttons operate identically to those described for the Main View window with the **All** option effect. Refer to "Primary Components of the Main View Window", page 1, for descriptions of these buttons.

Multiprocess Explorer Administrative Functions

The Admin menu in the **Multiprocess Explorer** window lets you perform several administrative functions. Only the **Save as Text**, **Close**, and **Exit** items are described here. All other options perform as those found in the Admin menu of the Main View window, described in "Admin Menu", page 6.

- **Save as Text**: The process status list from in the **Multiprocess Explorer** window is saved to the file you select using the Save Text dialog.

- **Launch Tool**: Allows you to run other WorkShop tools. You can switch to the other tools by selecting **Build Analyzer**, **Static Analyzer**, **Performance Analyzer**, or **Tester**. Selecting **Debugger** allows you to start another debugging session. If you have ProDev ProMP (formerly called WorkShop Pro MPF) installed on your system, the **Parallel Analyzer** selection is also available.

- **Remap Path**: Opens a dialog that allows you to enter a new pathname.

- **Close**: Closes the **Multiprocess Explorer** window only.

- **Exit**: Exits all views in the session and terminates the session.

Controlling Preferences

The **Preferences** option in the **Config** menu brings up the **Multiprocess Explorer Preferences** dialog that allows you to control when processes are added to the group and specifies their behavior. This option also contains a **Save** option that allows you to save your preferences. A sample configuration screen is shown below with typical settings for multiprocess debugging sessions.
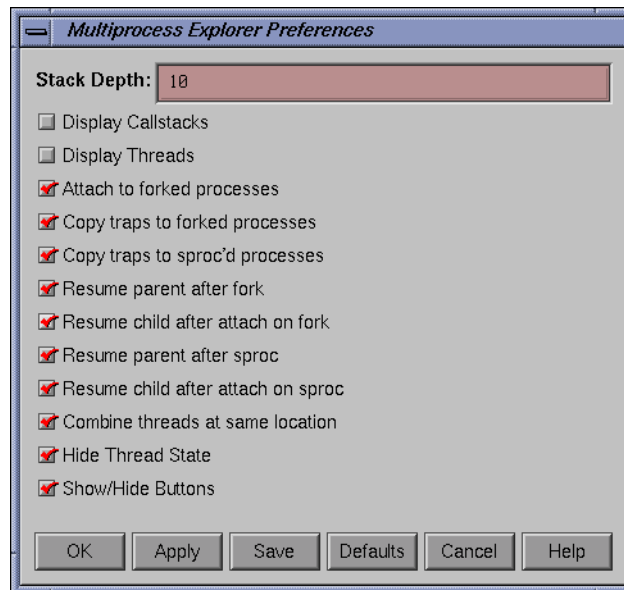

Figure 2-3 Multiprocess Explorer Preferences

The **Multiprocess Explorer** preference options are:

- **Stack Depth**: Allows you to set how many lines of the call stack should be displayed when opening the call stack. Default is 10.

- **Display Callstacks**: Allows you to specify that processes and threads in the Multiprocess Explorer window are opened to display their call stacks. This is useful if you have very large numbers of processes and/or threads and you would like to search all the call stacks for a particular name or address. The default is off.

- **Display threads**: Allows you to specify if the process node is always opened to display pthreads (if a process had pthreads). Default is off.

- **Attach to forked processes**: Automatically attaches new processes spawned by the `fork` command to the group. (Note that processes spawned by `sproc` are always attached.) Default is off.

- **Copy traps to forked processes**: Copies traps you have set in the parent process to new `forked` processes automatically. Alternatively, if you create parent traps with **Trap Manager** and specify `pgrp`, then the children inherit these traps automatically, regardless of the state of this flag. Default is off.

- **Copy traps to sproc'd processes**: Copies traps you have set in the parent process to new `sproc'd` processes automatically. As in the previous option, if you create parent traps with the **Trap Manager** and specify `pgrp`, the children inherit these traps automatically, whether this flag is set or not. Default is on.

- **Resume parent after fork**: Restarts the parent process automatically when a child is `forked`. Default is on.

- **Resume child after attach on fork**: Restarts the new `forked` process automatically when it is attached. If this option is left off, a new process stops as soon as it is attached. Default is on.

- **Resume parent after sproc**: Restarts the parent process automatically when a child is `sproc'd`. Default is on.

- **Resume child after attach on sproc**: Restarts the new `sproc'd` process automatically when it is attached. If this option is left off, a new process stops as soon as it is attached. Default is on.

- **Combine threads at same location**: Applies a collapsing algorithm to display threads stopped at the same location at the same time. (It is possible for threads to arrive at the same location through different logical routes.) Default is on.

- **Hide Thread State**: Displays thread state, which is the kernel-level status. Default is to hide the thread state.

- **Show/Hide Buttons**: When this option is **ON**, the **Continue All**, **Stop All**, **Step Into All**, **Step Over All**, **Sample All**, and **Kill All** buttons appear near the top of the **Multiprocess Explorer** window.

Controlling Multiple Processes

The **Process** menu of the **Multiprocess Explorer** allows you to control processes and threads. The **Process** menu has the following options:

Change focus to this entry

> Opens a dialog that allows you to switch the process or thread currently focused on in the Main View window to the process or thread selected in the **Multiprocess Explorer** window. Selecting a call stack entry changes the Main View window's focus to that process or thread and positions the **cvmain** window at the offset of the selected call stack.

Create a new window

> Brings up a new Main View window for the selected process or thread.

Goto

> Opens a dialog box that allows you to enter the name of a thread on which focus should be switched. This is useful when multiple threads, all at the same location, are collapsed into a single line. While **Change focus to this entry** always takes you to the first thread, **Goto** allows you to jump to any thread.

Add

> Opens a dialog in which you can select from a list of process ids. Selecting a process id (PID) in the dialog and pressing OK causes the process to be attached and added into the **Multiprocess Explorer** window's list of processes.

Remove

> After you select a process/thread (by highlighting it), click on **Remove** to remove it from the list of processes in the **Multiprocess**

Explorer window. A process in a sproc share group cannot be removed from the process group.

Many of these menu options are available via dynamic menus. Clicking and holding the right mouse button down shows the actions available on a popup menu. Quickly clicking down on the right mouse button gives the default action.

Source View

The **Source View** window is brought up by choosing **Views > Source View** from the Main View window menu bar. By default, a copy of the source on display in the Main View window source pane is displayed in this window.

The **Source View** menu bar contains selections duplicated from the Main View window: **Display**, **Traps**, and **Fix+Continue**. Each of these menus has the same functionality as its counterpart in the Main View window (see "Main View", page 105). The only new menu selection is the **File** menu described below:

- **Open**: Launches the Open dialog that allows you to choose a file to load into **Source View**.

- **Open Recent**: Lists recent files that have been opened previously.

- **Save**: Records changes made to the file during the current debugging session. You must first select **Make Editable** from this **File** menu when the file is read only.

- **Save As**: Records changes made during the debugging session to the source file under a different file name, the name of which you can enter or select using the **Save As Text** dialog.

- **Save As Text**: Records information in the display area as a text file, the name of which you can enter or select using the **Save As Text** dialog.

- **Open Separate**: Launches the **Open Separate** dialog that allows you to create a new **Source View** with the contents of a different source file.

- **Insert File**: Inserts the text of a file within your current file. This item description is available only if the file is editable. The **Make Editable** item description from this **File** menu can be used to switch the file from read only.

- **Clone**: Clones the current window.

- **Fork Editor**: Starts your default editor on the current file. The default editor is determined by the `editorCommand` resource in the `app-defaults` file. The value of this resource defaults to `wsh -c vi +%d`, which means run `vi` in a `wsh` window and scroll to the current line. If the editor allows you to specify a starting line, enter `%d` in the resource to indicate the new line number.

- **Recompile**: Displays the **Build View** window that allows you to compile the source code associated with the current executable.

- **Make Editable**: Toggles the source code displayed between **(Read Only)** and **(Editable)** so that you can edit your code.

- **Search**: Searches for a literal case-sensitive, literal case-insensitive, or regular expression. After you have set your target and clicked **Apply** (or pressed `Enter`), each instance is marked by a search target indicator in the scroll bar. You can search forward or backward in the file by clicking the **Next** or the **Prev** button. You can also click an indicator with the middle mouse button to scroll to that point. Clicking **Reset** removes the search target indicators.

- **Go To Line**: Launches the **Go To Line** dialog that allows you to go to a specific line in the source. You can type in the line, or select the line number via the slider bar.

- **Versioning**: Provides access to the configuration management tool, if you have designated one.

  The `cvconfig` script allows you to designate ClearCase, RCS or SCCS. Type the following:

  /usr/sbin/cvconfig [rcs | sccs | ptools | clearcase]

  You must have `root` permissions to run `cvconfig`.

  Selecting any option displays a shell in which you can access the configuration management tool. The selections in the submenu are:

  – **CheckIn** — Saves the source file and checks it into the database as a new version.

  – **CheckOut** — Recalls the source file from the tool's database if you have the proper authority, locks it, and makes it editable.

  – **UncheckOut** — Cancels the checkout, with no changes registered.

- **Close**: Dismisses the **Source View** window.

Process Meter

The **Process Meter** window is brought up by choosing **Views > Process Meter** from the Main View menu bar. The **Process Meter** monitors resource usage of a running process without saving the data. Figure 2-4, page 28, shows the Process Meter in its default configuration (with only the **User Time** and **Sys Time** fields active).
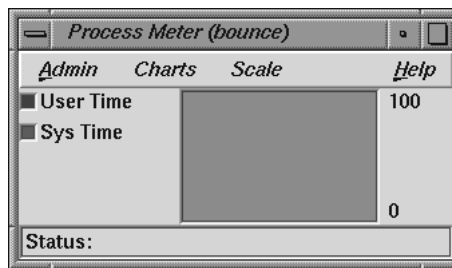
Figure 2-4 Process Meter

The **Process Meter** contains its own menu bar that contains the **Admin**, **Charts**, **Scale**, and **Help** menus. The **Admin** menu is the same as that described in "Admin Menu, Task View", page 29. The **Help** menu is the same as that described in "Help Menu", page 17. The other menus are described in the following list:

- **Charts** Menu: The **Charts** menu contains a set of toggles that allow you to choose which charts are displayed in the **Process Meter** window. You can display as many charts simultaneously as you wish. The following choices are available:

    User/Sys Time (the default)
    Major/Minor Faults
    Context Switches
    Bytes Read/Written
    Read/Write Sys Calls
    Other Sys Calls
    Total Sys Calls
    Signals
    Process Size

- **Scale** Menu: The **Scale** menu allows you to set the time scale for the processes displayed in the **Process Meter** window. A menu allows you to choose a time scale from 2 seconds to 10 minutes.

Ada-specific Windows

This chapter discusses the **Task View** and **Exception View** windows that are specific to Ada code.

Task View

Select **Views > Task View** from the Main View window menu bar to call up the **Task View** window. The **Task View** window is an Ada-specific view that provides you with task and call stack information. If you do not have Ada installed on your system, the **Task View** menu option of the **Views** menu is grayed out.

The **Task View** menu bar contains the **Admin**, **Config**, **Layout**, **Display**, and **Help** menus. The **Help** menu is the same as that described in "Help Menu", page 17. Other menus are described in the following sections.

Admin Menu, Task View

The **Admin** menu contains the following options:

- **Active**: This toggle activates the current window in a set of cloned windows.

- **Clone**: Creates a clone of the current window. This action is not supported in the current release, and the option is grayed out.

- **Save As Text**: Launches the **Save Text** dialog. This dialog allows you to save your current session as text in a file you designate.

- **Close**: Closes the current window.

Config Menu, Task View

The **Config** menu contains the **Preferences** option, which launches the **Task View Preference** dialog that allows you to set maximum call stack depth shown in **Task View**. Default depth is 32 frames.

Layout Menu, Task View

The **Layout** menu contains the following toggles:

- **Task List**: Causes only the **CallStack View** to be shown.

- **Single Task**: Causes only the **Process Display** to be shown.

Display Menu, Task View

The **Display** menu is divided into the **Task List Format** and **Callstack Format** sections. The **Task List Format** toggle buttons control which buttons appear in the toggle sort list, as well as what information is displayed in the **Process Display** area. The **Callstack Format** toggles control the amount of information to be displayed in the **Callstack Display** area of the**Task View** window.

The **Task View** Display menu contains the following toggles:

- **Thread/task**: Displays thread/task number. This toggle is active by default.

- **Status**: Displays process status. This toggle is active by default.

- **PID**: Displays PID number.

- **Location**: Displays routine name and location in the current source file.

- **Arg Values**: Allows you to set the argument values in the **Callback Display**. This toggle is active by default.

- **Arg Names**: Allows you to set the argument names in the **Callback Display**. This toggle is active by default.

- **Arg Types**: Allows you to display argument types in the **Callback Display**.

- **PC**: Allows you to set the program counter (PC) in **Task View**.

In addition to menus, **Task View** also contains the following items from which you can select to vary the display:

- Sort toggles: Allows you to sort the process list by **Thread**, **Name**, **State**, **Pid**, or **Location**, depending on which of the buttons is active. Default selection is **Thread**.

- Process display tabs: Allows you to view a list of tasks or details of the currently running (highlighted) task.

- Callstack display tabs: Allows you to view all call stack information or call stack details of the currently selected process.

Exception View

Select **Views > Exception View** from the Main View window menu bar to display the **Exception View** window. **Exception View** is an Ada-specific view that allows you to set traps on exceptions and control exception handling. This view works only if the Ada compiler is installed. By default, this view displays only the following predefined Ada exceptions:

- Constraint errors

- Program errors

- Storage errors

- Tasking errors

In addition, a single breakpoint is set on any unhandled exception.

The **Stop** boxes toggle on and off to indicate whether a trap is active.

The **When** control menus allow you to determine when an exception trap fires. The following choices are available:

- **Always**: Stops any time the exception is raised.

- **WhenOthers**: Stops when caught by a `when others` handler rather than an explicit handler or when unhandled.

- **Unhandled**: Stops when the exception is unhandled.

In the un-labeled text field at the bottom right of the window you can enter a single, fully qualified Ada exception name or a single, fully qualified Ada unit name. Depending on whether the **add**, **remove**, or **find** mode is active; pressing Enter causes one of the following actions to occur:

- **add** mode:

    - Single exception: Adds single exception to the exception list

    - Library unit name: Adds all exceptions found in that library unit name to the exception list

- **remove** mode:

  – Single exception: Removes single exception from the exception list

  – Library unit name: removes all exceptions found in that library unit name from the exception list

- **set** mode

- **clear** mode

- **find** mode:

  – Single exception: positions top of the exception list to single exception

  – Library unit name: positions top of the exception list to the first exception found in given library unit name

Admin Menu, Exception View

The **Admin** menu has the following options:

- **Active**: Activates the current window in a set of cloned windows.

- **Clone**: Creates a clone of the current window.

- **Save As Text**: Launches the **Save Text** dialog. This dialog allows you to save your current session as text in a file you designate.

- **Close**: Closes the current window.

Config Menu, Exception View

The **Config** menu has the following options:

- **Load Exceptions**: Opens the **Load User Defined Exceptions** dialog that allows you to add additional exceptions to the predefined Ada exceptions.

- **Save Exceptions**: Opens the **Save User Defined Exceptions** dialog that allows you to save any user-defined exceptions to the predefined Ada exceptions.

Display Menu, Exception View

The **Display** menu has the following options:

- **Delete All**: Deletes all exception traps.

- **Clear All Traps**: Clears all exception traps. Clearing traps is not the same as deleting traps. Clearing only temporarily affects traps while deleting removes them permanently.

- **Reset All Buttons**: Resets all button actions.

X/Motif Analyzer Windows

The X/Motif Analyzer provides specific debugging support for X/Motif applications. There are various examiners for different X/Motif objects, such as widgets and X Window System graphics context, that might be difficult or impossible to inspect using ordinary debugger functionality. See *ProDev WorkShop: Debugger User's Guide* for a comprehensive discussion and tutorial regarding the X/Motif Analyzer.

To access the **X/Motif Analyzer** window, pull down the **Views** menu and select **X/Motif Analyzer**.

Global Objects

Though the X/Motif Analyzer is made up of several different examiner windows, a number of objects, such as the **Admin** menu, **Examine** menu, **Help** menu and several text bars, remain constant throughout window changes. The following examiners are available and discussed:

- "Breakpoints Examiner", page 38
- "Trace Examiner", page 47
- "Widget Examiner", page 49
- "Tree Examiner", page 49
- "Callback Examiner", page 52
- "Window Examiner", page 52
- "Event Examiner", page 52
- "Graphics Context Examiner", page 52
- "Pixmap Examiner", page 53
- "Widget Class Examiner", page 53

Admin Menu

The **Admin** menu offers the following menu selections:

- **Active**: Activates the current window in a set of cloned windows. In the current release, this toggle is always active.

- **Clone**: Creates a clone of the current window. This action is not supported in the current release and the option is grayed out.

- **Save As Text**: Launches the **Save Text** dialog. This dialog allows you to save your current session as text in a file you designate. This selection is not available for examiners that are graphical displays, such as the Breakpoints Examiner, the Tree Examiner, and the Pixmap Examiner.

- **Close**: Closes the current window.

Examine Menu

The **Examine** menu offers the following options:

- **Selection**: Selects the currently highlighted object for examination. You must first highlight the name of an object before you select this option.

- **Widget**: Uses the current selection as input to the widget examiner, then opens that examiner (see "Widget Examiner", page 49, for information).

- **Widget Tree**: Switches the window view to the widget tree examiner (see "Tree Examiner", page 49, for information).

- **Widget Class**: Switches the window view to the widget class examiner (see "Widget Class Examiner", page 53, for information).

- **Window**: Switches the window view to the window examiner (see "Window Examiner", page 52, for information).

- **X Event**: Switches the window view to the X Event examiner (see "Event Examiner", page 52, for information).

- **X Graphics Context**: Switches the window view to the X graphics context examiner (see "Graphics Context Examiner", page 52, for information).

- **X Pixmap**: Switches the window view to the X pixmap examiner (see "Pixmap Examiner", page 53, for information).

Examiner Tabs

In addition to access through the **Examine** menu, each examiner can be accessed through a tab at the bottom of each view (see Figure 4-1).

Figure 4-1 Examiner Tabs

When first launched, the X/Motif Analyzer has the following four tabs from left-to-right:

- **Breakpoints**
- **Trace**
- **Widget**
- **Tree**

As you select other examiners through the **Examine** menu, new tabs are added for the new examiners.

To delete a tab:

1. Select the tab you want to delete.

2. Right-click and select **Remove Examiner** from the pop-up menu.

   The selected tab disappears.

   Note: You can not remove the first four tabs.

The X/Motif Analyzer also brings up new examiner windows whenever they are needed. Click on the collapsed tabs to the right to display them.

Return Button

Both the **Widget** and **Name** text fields have return buttons (see Figure 4-2, page 39) just to the right. Clicking these buttons causes the X/Motif Analyzer to respond

exactly as if you had pressed Return on your keyboard.Only the **Breakpoint Examiner** and **Widget Examiner** have a **Return** button.

Breakpoints Examiner

The Breakpoints examiner is not really an examiner, but a control area where you can set widget-level breakpoints. The breakpoints examiner is divided into three areas (see Figure 4-2, page 39):

- The widget specification area that contains the same information as that in the Widget examiner. You can select a widget address, name, or class in this area, as well as move to the widgets parents or children, or select a widget in the application. In cases where the breakpoint type does not apply to widgets (for example, input-handler breakpoints), this area is blank.

- The parameter specification area, the contents of which vary according to the type of breakpoint you are setting. For example, for callback breakpoints, this area contains the callback name and client data; for event-handler breakpoints, it contains the event type and the client data, and so on.

- The breakpoint area, which contains the breakpoint name, a search field, and the **Add**, **Modify**, **Delete**, and **Step To** buttons. Because the **Search** text field and the four buttons appearing in the **Breakpoints** area function the same way no matter which **Breakpoint Type** is selected, descriptions for these items are included here, and are not described in each of the remaining subsections. These are:

  - **Search** text field: Allows you to perform a text search through your breakpoints.

  - **Add** button: Allows you to add a new breakpoint.

  - **Modify** button: Allows you to change the selected breakpoint's settings.

  - **Delete** button: Deletes the selected breakpoint.

  - **Step To** button: Allows you to step to the next condition. **Step To** creates a temporary breakpoint, resumes the process, and waits until the process stops. This temporary breakpoint acts exactly like an ordinary breakpoint, save that the **Step To** button automatically resumes the process and puts up a busy cursor until the condition becomes true.
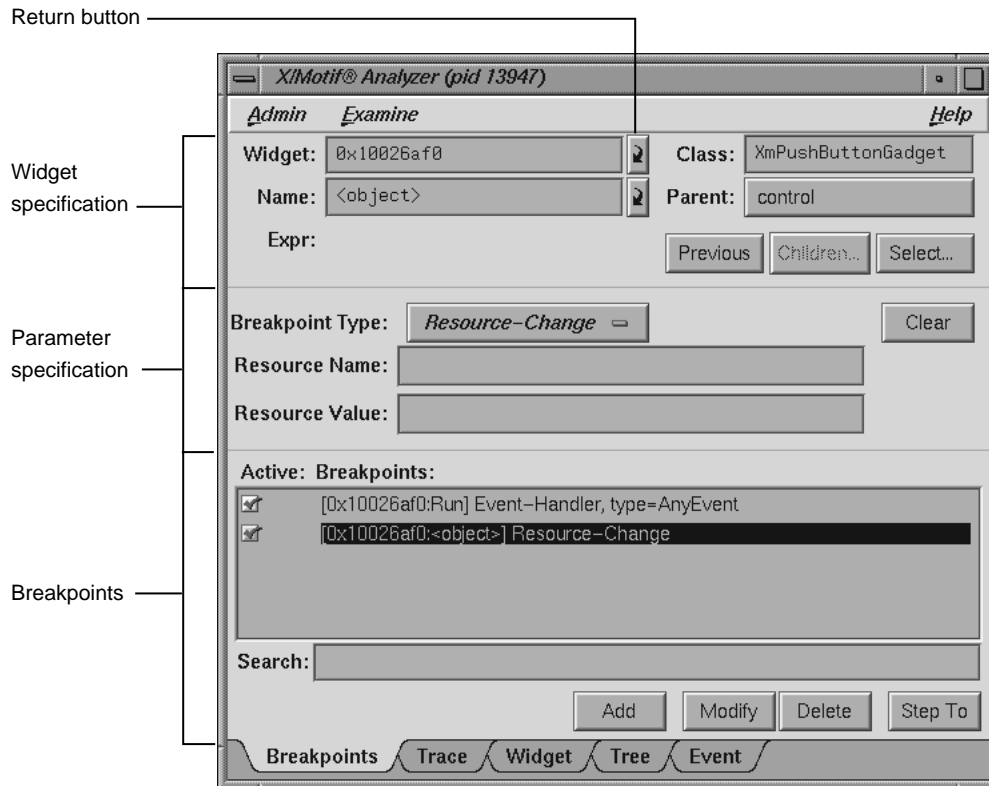
Figure 4-2 Breakpoints Examiner Display in the **X/Motif Analyzer** Window

The control area has eight different breakpoint types that it can examine. These types are set through the **Breakpoint Type** options. The following **Breakpoint Type** options are available:

- **Callback**: Widget callback installed by `XtAddCallback`. Parameters include callback name and `client_data XtPointer` value. See "Callback Breakpoints Examiner", page 40, for more information.

- **Event-Handler**: Widget event handler installed by `XtAddEventHandler`. Parameters include X event type and `client_data XtPointer` value. See "Event-Handler Breakpoints Examiner", page 41, for more information.

- **Resource-Change**: Resource change caused by `XtSetValues` or `XtVaSetValues`. Parameters include resource name and resource value, both strings. See "Resource-Change Breakpoints Examiner", page 42, for more information.

- **Timeout-Procedure**: Timeout callback installed by `XtAppAddTimeOut`. Parameters include `client_data XtPointer` value. See "Timeout-Procedure Breakpoints Examiner", page 42, for more information.

- **Input-Handler**: Input callback installed by `XtAppAddInput`. Parameters include `client_data XtPointer` value. See "Input-Handler Breakpoints Examiner", page 43, for more information.

- **State-Change**: Various widget state changes (for example, managed or realized). Parameters include widget state. See "State-Change Breakpoints Examiner", page 44, for more information.

- **X-Event**: X event received by target application. Parameters include X event type. See "X-Event Breakpoints Examiner", page 45, for more information.

- **X-Request**: X request received by target application. Parameters include X request type. See "X-Request Breakpoints Examiner", page 46, for more information.

Callback Breakpoints Examiner

When the **Callback** option of the **Breakpoint Type** option button in the **Breakpoints Examiner** is selected, the Callback Breakpoints Examiner is displayed.

The Callback Breakpoints examiner contains the following items:

- **Widget** text field: Allows you to choose a widget to examine by entering the widget address.

- **Name** text field: Allows you to choose a widget to examine by entering the widget name.

- **Class** text field: Allows you to choose a widget to examine by entering the widget's class. Leave the field blank or enter **Any** to select all widgets.

- **Parent** text field: Allows you to move to the parent of the currently selected widget.

- **Previous** button: Moves you to the previously selected widget.

- **Children** button: Shows you the widget's children (it is grayed out if the selected widget cannot have children).

- **Select** button: Allows you to select the widget in the target process.

- **Breakpoint Type** option button: Allows you to select the type of breakpoint you wish to set. In this section, **Callback** is selected.

- **Clear** button: Clears all the current breakpoint selections and text fields.

- **Callback Name** text field: Allows you to set the **Name** of the callback for the breakpoint.

- **Client_Data** text field: Allows you to pass and get back pointer values for **Client_Data**.

Event-Handler Breakpoints Examiner

When the **Event-Handler** option of the **Breakpoint Type** option button in the Breakpoint Examiner is selected, then Event-Handler examiner appears.

The Event-Handler Breakpoints examiner contains the following items:

- **Widget** text field: Allows you to choose a widget to examine by entering the widget address.

- **Name** text field: Allows you to choose a widget to examine by entering the widget name.

- **Class** text field: Allows you to choose a widget to examine by entering the widget's class. Leave the field blank or enter **Any** to select all widgets.

- **Parent** text field: Allows you to move the parent of the currently selected widget.

- **Previous** button; Moves you to the previously selected widget.

- **Children** button: Shows you the widget's children (it is grayed out if the selected widget cannot have children).

- **Select** button: Allows you to select the widget in the target process.

- **Breakpoint Type** option button: Allows you to select the type of breakpoint you wish to set. In this section, **Event-Handler** is selected.

- **Clear** button: Clears all the current breakpoint selections and text fields.

- **Event Type** option button: Allows you to set the event type for a given breakpoint.

- **Client_Data** text field: Allows you to pass and get back pointer values for the **Client_Data**.

Resource-Change Breakpoints Examiner

When the **Resource-Change** option of the **Breakpoint Type** option button in the Breakpoint Examiner is selected, the examiner appears.

The Resource-Change Breakpoints examiner contains the following items:

- **Widget** text field: Allows you to choose a widget to examine by entering the widget address.

- **Name** text field: Allows you to choose a widget to examine by entering the widget name.

- **Class** text field: Allows you to choose a widget to examine by entering the widget's class. Leave the field blank or enter **Any** to select all widgets.

- **Parent** text field: Allows you to move the parent of the currently selected widget.

- **Previous** button: Moves you to the previously selected widget.

- **Children** button: Shows you the widget's children (it is grayed out if the selected widget cannot have children).

- **Select** button: Allows you to select the widget in the target process.

- **Breakpoint Type** option button: Allows you to select the type of breakpoint you wish to set. In this section, **Resource-Change** is selected.

- **Clear** button: Clears all the current breakpoint selections and text fields.

- **Resource Name** text field: Allows you to set the resource name for the breakpoint.

- **Resource Value** text field: Allows you to set the resource value for the breakpoint.

Timeout-Procedure Breakpoints Examiner

When the **Timeout Procedure** option of the **Breakpoint Type** option button in the Breakpoint Examiner is selected, the examiner appears as shown in Figure 4-3.
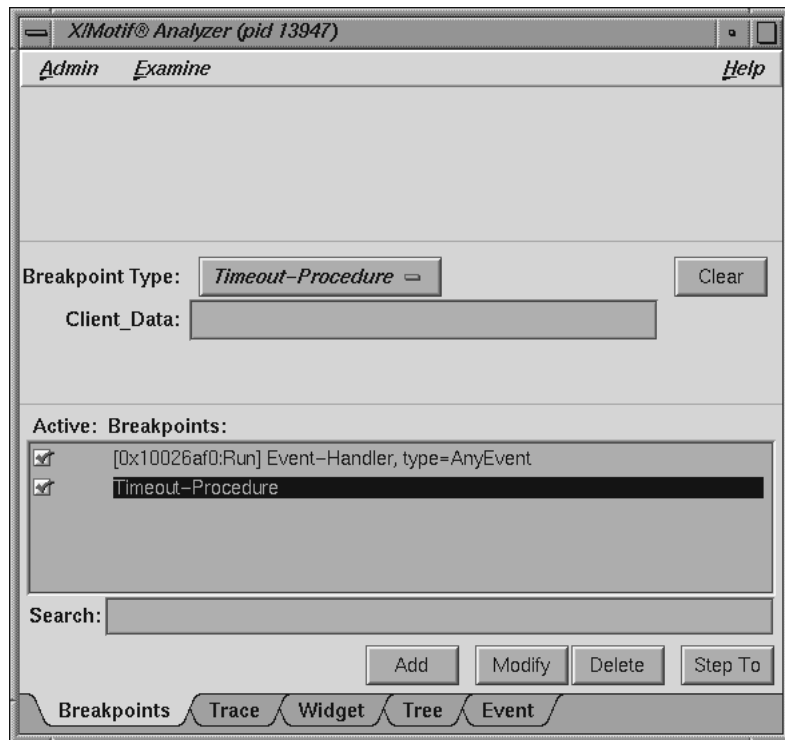
Figure 4-3 Timeout-Procedure Breakpoints Examiner

The Timeout-Procedure Breakpoints examiner contains the following items:

- **Breakpoint Type** option button: Allows you to select the type of breakpoint you wish to set. In this section, **Timeout-Procedure** is selected.

- **Clear** button: Clears all the current breakpoint selections and text fields.

- **Client_Data** text field: Allows you to pass in and get back pointer values for the **Client_Data**.

Input-Handler Breakpoints Examiner

When the **Input-Handler** option of the **Breakpoint Type** option button in the Breakpoint Examiner is selected, the examiner appears.

The Input-Handler Breakpoints examiner contains the following items:

- **Breakpoint Type** option button: Allows you to select the type of breakpoint you wish to set. In this section, **Input-Handler** is selected.

- **Clear** button: Clears all the current breakpoint selections and text fields.

- **Client_Data** text field: Allows you to pass in and get back pointer values for the **Client_Data**.

State-Change Breakpoints Examiner

When the **State-Change** option of the **Breakpoint Type** option button in the Breakpoint Examiner is selected, the examiner appears.

The State-Change Breakpoints examiner contains the following items:

- **Widget** text field: Allows you to choose a widget to examine by entering the widget address.

- **Name** text field: Allows you to choose a widget to examine by entering the widget name.

- **Class** text field: Allows you to choose a widget to examine by entering the widget's class. Leave the field blank or enter **Any** to select all widgets.

- **Parent** button: Allows you to move the parent of the currently selected widget.

- **Previous** button: Moves you to the previously selected widget.

- **Children** button: Shows you the widget's children (it is grayed out if the selected widget cannot have children).

- **Select** button: Allows you to select the widget in the target process.

- **Breakpoint Type** option button: Allows you to select the type of breakpoint you wish to set. In this section, **State-Change** is selected.

- **Clear** button: Clears all the current breakpoint selections and text fields.

- **State Type** option button: Allows you to set the state change type for a given breakpoint.

X-Event Breakpoints Examiner

When you select the **X-Event** option of the **Breakpoint Type** option button in the Breakpoint Examiner, the examiner appears as shown in Figure 4-4.
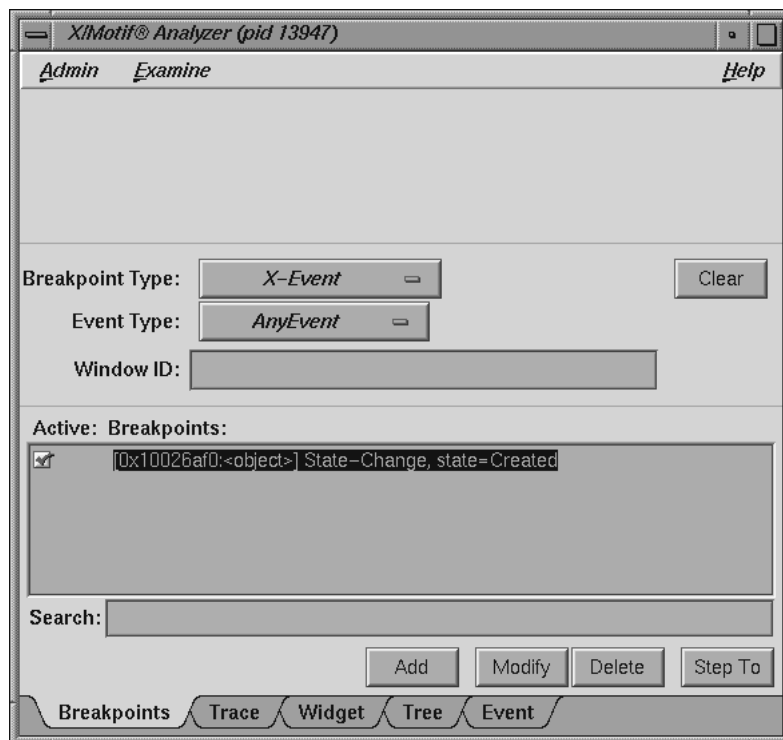
Figure 4-4 X-Event Breakpoints Examiner

The X-Event Breakpoints examiner contains the following items:

- **Breakpoint Type** button: Allows you to select the type of breakpoint you wish to set. In this section, **X-Event** is selected.

- **Clear** button: Clears all the current breakpoint selections and text fields.

- **Event Type** button: Allows you to set the event type for a given breakpoint.

- **Window ID** text field: Allows you to set the Window ID value for the breakpoint.

X-Request Breakpoints Examiner

When the **X-Request** option of the **Breakpoint Type** option button in the Breakpoint Examiner is selected, the examiner appears.

The X-Request Breakpoints examiner contains the following items:

- **Breakpoint Type** option button: Allows you to select the type of breakpoint you wish to set. In this section, **X-Request** is selected.

- **Clear** button: Clears all the current breakpoint selections and text fields.

- **Request Type** button: Launches the **Request Type Selection** dialog (see Figure 4-5). This dialog allows you to select the type of X-Request used for your breakpoint. The information displayed is in outline form; selecting a given item selects all its subitems. For example, if you select `Window-Category`, `CreateWindow`, `ChangeWindowAttributes`, `GetWindowAttributes`, and so on are also selected.
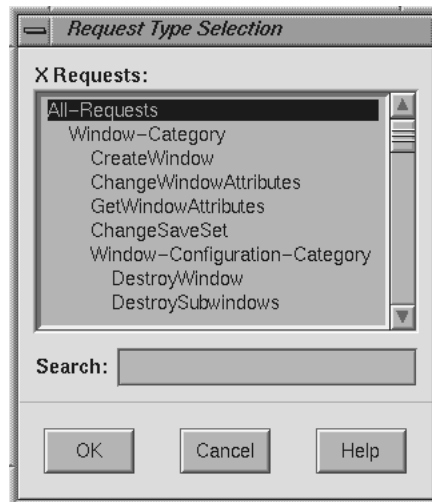
Figure 4-5 Request Type Selection Dialog

Trace Examiner

The Trace examiner (see Figure 4-6, page 48) is a control area where you can trace the execution of your application and collect the following date:

- X Server Events

- X Server Requests

- Widget Event Dispatch Information

- Widget Resource Changes (through `XtSetValues`)

- Widget State Changes (`create`, `destroy`, `manage`, `realize`, and `unmanage`)

- **Xt** Callbacks (`widget`, `event handler`, `work proc`, `timeout`, `input`, and `signal`)
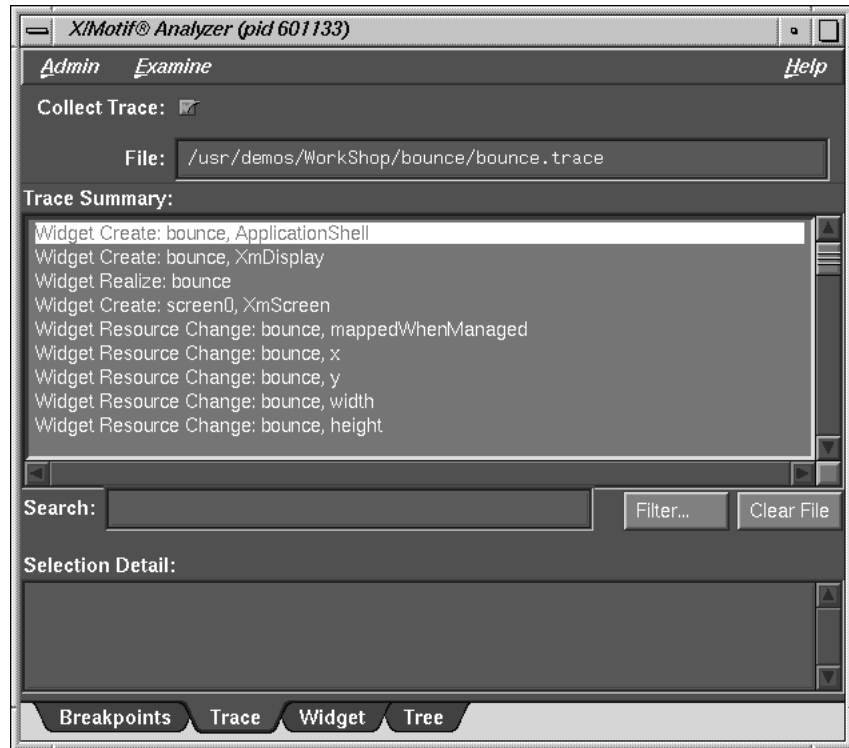
Figure 4-6 Trace Examiner

The Trace examiner contains the following items:

- **Collect Trace** toggle: Allows you to turn the tracing on and off.

- **File** text field: Allows you to select the file name for the trace. If no file is selected, a default *filename* for the trace is chosen.

- **Search** text field: Allows you to perform an incremental, textural search for the trace list.

- **Filter** button: Launches a dialog that allows you to select the trace entry types you want displayed in the list.

- **Clear File** button: Erases the trace file. Any subsequent trace information goes to the beginning of the file.

Widget Examiner

The Widget examiner displays the internal Xt widget structure, as well as the Xt inheritance implementation using nested C constructs.

The Widget examiner contains the following items:

- **Widget** text field; Allows you to choose a widget to examine by entering the widget address.

- **Name** text field: Allows you to choose a widget to examine by entering the widget name.

- **Parent button**: Allows you to move the parent of the currently selected widget.

- **Previous** button: Moves you to the previously selected widget.

- **Children** button: Shows you the widget's children. (It is grayed out if the selected widget does not have children.)

- **Select** button: Allows you to select the widget in the target process.

Tree Examiner

The Tree examiner (see Figure 4-7) displays the widget hierarchy.
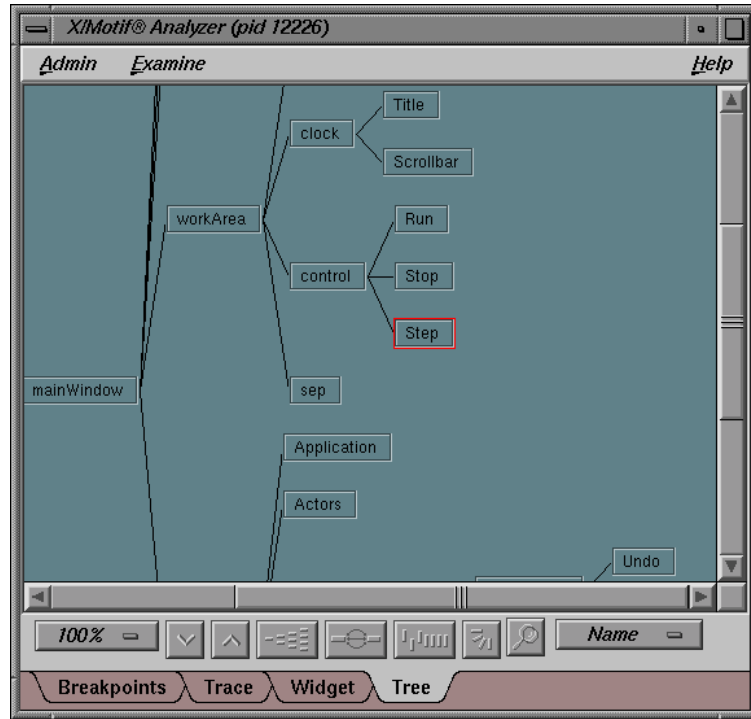
Figure 4-7 Tree Examiner

You can double-click a node to view that widget in the Widget examiner.

If the Tree examiner is currently selected, it does not automatically fetch the current widget tree each time the process stops. To force retrieval of the widget tree, select another examiner and then go back to the Tree examiner. Or, click on the **Tree** tab.

The graphical buttons across the bottom of the Tree Examiner window from left-to-right have the following functions.
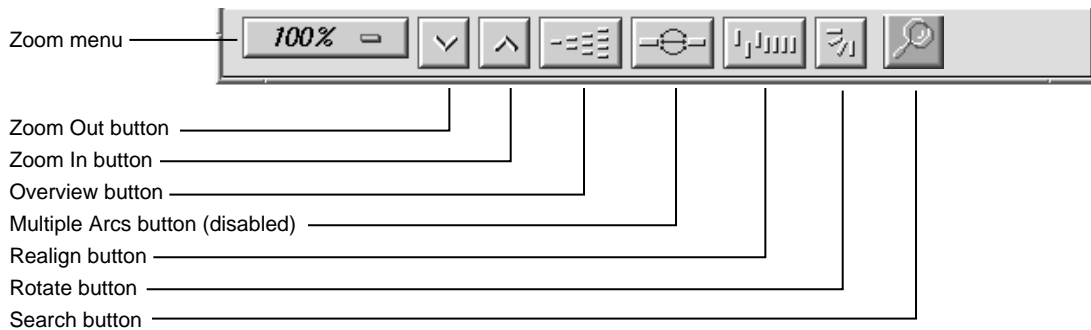
Figure 4-8 Tree Examiner Window Graphical Buttons

- Zoom percentage indicator: Click on this indicator to bring up a submenu from which you can select a different percentage.

- Zoom Out button: Decreases the zoom percentage.

- Zoom In button: Increases the zoom percentage.

- Overview button: Shows the entire tree structure in another window along with an indication of which portion of the tree is currently being displayed in the full-sized Tree Examiner window.

- Multiple Arcs button: This button shows/hides multiple connections between nodes on the graph. For example, if main calls foo several times, you see a line (arc) for all calls made.

- Realign button: This button resets the graph back to its original configuration (before you began working with it). Also, hidden nodes reappear.

- Rotate button: Change the format of the window from up-down to left-right, or vice-versa.

- Search button: Search for text in the tree using a Widget Hierarchy Search Text window.

- Widget View Type Option button: Click on this indicator to change the kind of information being displayed in the nodes of the tree. This brings up a submenu which allows you to choose Name, Class, ID (widget address) or Window (indicate <gadget> or address).

Callback Examiner

The Callback examiner automatically appears when the process is stopped somewhere in a callback. It first displays the callstack frame. Then it displays information about the widget in the callback. Finally, it displays the proper callback structure contained in the **call_data** argument to the callback procedure, based on the widget type and the callback name.

Window Examiner

The Window examiner displays window attributes for an X window and the parent and children window IDs. These attributes are returned by `XGetWindowAttributes`, with decoding of the visual structure, enums, and masks.

The Window examiner contains the **Window** text field that displays the address of the window that is being examined. You may change to a different window by entering a new address and pressing Enter.

Event Examiner

The Event examiner displays the event structure for an **XEvent** pointer. The proper **XEvent** union member is used, and enums and masks are decoded.

The Event examiner contains the **X Event** text field, which displays the address of the X event that is being examined. You may change to a different X event by entering a new address and pressing Enter.

Graphics Context Examiner

The Graphics Context examiner displays the X graphics context attributes that are cached by `Xlib` in the form of an `XGCValues` structure. Enums and masks are decoded.

The Graphics Context examiner contains the **GC** text field that displays the address of the graphics context that is being examined. You may change to a different context by entering a new address and pressing Enter.

Pixmap Examiner

The Pixmap examiner displays basic attributes of an X pixmap, like size and depth. It also attempts to provide an ASCII display of small pixmaps, using the units digit of the pixel values.

The Pixmap examiner displays the contents of an X pixmap. To specify an X pixmap identifier, enter a numeric expression in the top text field of the window. Then, use `default` as the colormap identifier to specify the default X colormap for your screen. In the pixmap display, left-click on a pixel to see the pixel value, position, and red-green-blue intensities.

Widget Class Examiner

The Widget Class examiner displays the `Xt` widget class structure, as well as the `Xt` inheritance implementation using nested C constructs.

The Widget Class examiner contains the **W Class** text field, which displays the address of the widget class that is being examined. You may change to a different widget class by entering a new address and pressing `Enter`.

Managing Traps

In addition to setting traps by using the command line, the **Views** menu of the Main View window provides you with three views specific to trap management:

- **Trap Manager**

- **Signal Panel**

- **Syscall Panel**

Call up the **Trap Manager** window from the Main View window menu bar as follows by selecting **Trap Manager** from the **Views** menu.

Trap Manager

The **Trap Manager** allows you to set, edit, and manage traps (used in both the Debugger and Performance Analyzer).

The **Trap Manager** window contains the following items (besides the menu bar, which is discussed below):

- **Trap** text field: Contains a description of the trap.

- **Condition** text field: Contains the condition of the trap.

- **Cycle Count** text field: Displays the current cycle count.

- **Current Count** text field: Displays the current trap count.

- **Full** button: Allows you to toggle between display of full and partial path names.

- **Modify** button: Allows you to change the selected breakpoint's settings.

- **Add** button: Allows you to add a new breakpoint.

- **Clear** button: Clears all the current breakpoint selections and text fields.

- **Delete** button: Deletes the selected breakpoint.

- **Active** label: If selected with a check mark, the trap is enabled.

- **Trap** display area: Contains a description of each trap, and a toggle to indicate whether or not the trap is active.

- **Search** text field: Allows you to perform an incremental textual search for the trap list.

The **Trap Manager** window has a menu bar which contains the **Admin**, **Config**, **Traps**, **Display**, and **Help** menus. The **Admin** menu is the same as that described in "Admin Menu, Task View", page 29. The **Help** menu is the same as that described in "Help Menu", page 17. The other menus are described in the following list:

- The **Config** menu contains the following items:

  - **Load Traps**: Brings up the **File** dialog allowing you to load the traps from a file.

  - **Save Traps**: Brings up the **File** dialog allowing you to save the current traps to a file.

- The **Traps** menu has options that allow you to set traps under a number of conditions. The following conditions are available:

  - **At Source Line**: Highlight a line in the Main View window's source pane before selecting this option to set a breakpoint at the selected line.

  - **Entry Function**: Highlight a function name in the Main View window's source pane before selecting this option to set a breakpoint at the entry to the function.

  - **Exit Function**: Highlight a function name in the Main View window's source pane before selection this option to set a breakpoint at the exit from the function.

  - **Stop Trap Default**: Causes a trap created to be a "stop" trap. Toggles with the **Sample Trap Default**.

  - **Sample Trap Default**: Causes a trap created to be a "sample" trap. Toggles with the **Stop Trap Default**.

  - **Group Trap Default**: ON/OFF toggle to cause a trap created to have the "pgrp" option.

  - **Stop All Default**: ON/OFF toggle to cause a trap created to have the **All** option.

- The **Display** menu contains the **Delete All** optoin, which deletes all traps from the trap list.

Signal Panel

The **Signal Panel** displays the signals that can occur. You can specify which signals trigger traps and which are to be ignored.

The **Signal Panel** contains an **Admin** menu (described in "Admin Menu, Task View", page 29) and a **Help** menu (described in "Help Menu", page 17). Each signal trigger trap in the display has a toggle associated with it. In addition, the panel has a **Search** text field.

Note: When debugging IRIX 6.5 pthreads, the **Signal Panel** is inaccessible if more than one thread is active.

Syscall Panel

The **Syscall Panel** allows you to set traps at the entry to or exit from system calls.

The **Syscall Panel** contains an **Admin** menu (described in "Admin Menu, Task View", page 29) and a **Help** menu (described in "Help Menu", page 17). Each system call in the display has two toggles associated with it: one to set a trap on entry, one to set a trap on exit. In addition, the panel has a **Search** text field.

Data Examination Windows

There are several windows that are used primarily to examine your program's data:

- "Array Browser Window", page 59

- "Call Stack Window", page 72

- "Data View Window", page 73

- "Expression View Window", page 73

- "File Browser Window", page 75

- "Data Explorer Window", page 76

- "Variable Browser Window", page 86

Array Browser Window

To examine numeric, pointer, or character string data in an array variable, select **Array Browser** from the **Views** menu at a point in the process where the variable is present. The **Array Browser** allows you to view elements in a multi-dimensional array (up to 100 x 100 elements), presented in a spreadsheet and graphically, if desired.
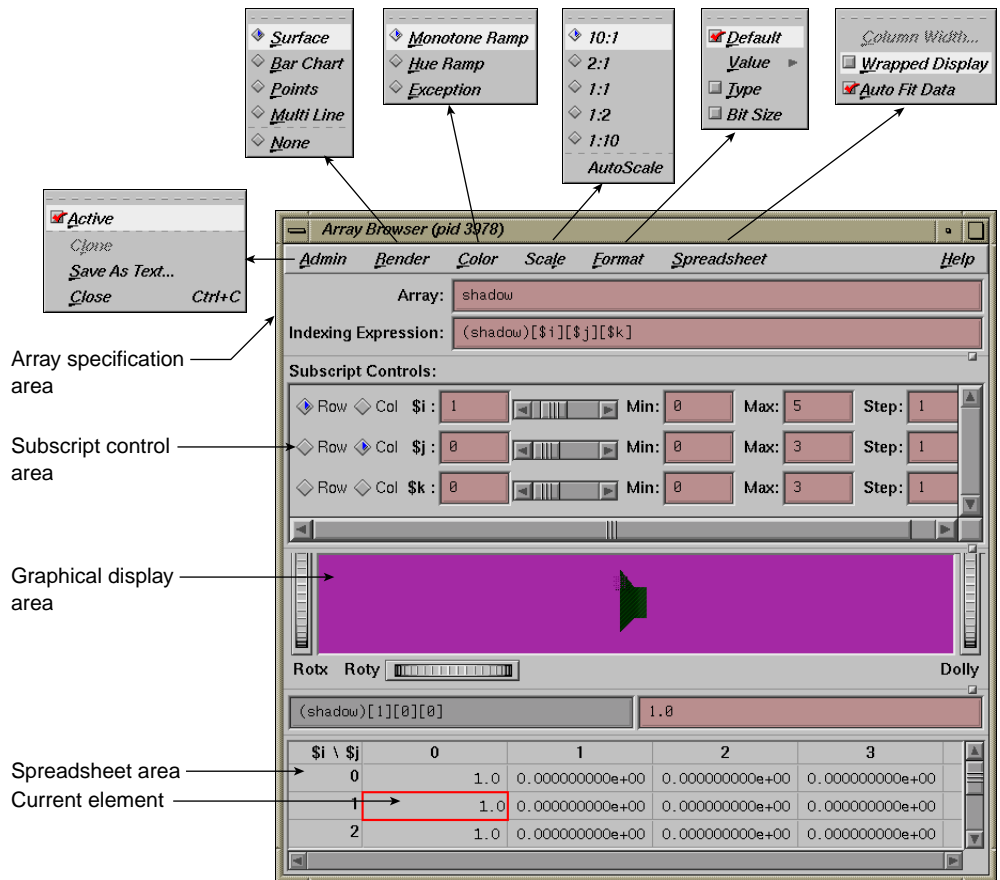
Figure 6-1 **Array Browser** with **Display** Menu Options

Note: The **Render**, **Color**, and **Scale** tear-off menus are available only if you are have an SGI workstation with Open Inventor installed on it.

The *array specification area* allows you to specify the variable and its dimensions. It consists of the following fields:

- **Array** : Allows you to enter the name of the array variable. This entry is language-dependent.

For Fortran, the expression may be an array or a dummy array variable name. If the last dimension of the array is unspecified (*), a subscript value of 1 is assumed initially.

For C and C++, the entry may be an array or a pointer. If pointers are used, the expression is treated as though it were a single element, in which case you need to use the subscript controls to see more than the first element.

- **Indexing Expression** : The expression used to view an element in the array. It is filled in automatically when you specify an array to view.

  The expression supplied is language-specific. It represents the indexing expression used in the language to access a particular element. The subscripts are specified by special indexing variables ($i, $j, $k, and so forth) that can be manipulated in the subscript controls area.

The **Subscript Controls** area allows you to control which elements in the variable are displayed and allows you to shift the current element. The number of dimensions in the array governs the number of controls that are displayed. A close-up view of the subscript controls area appears in Figure 6-2.
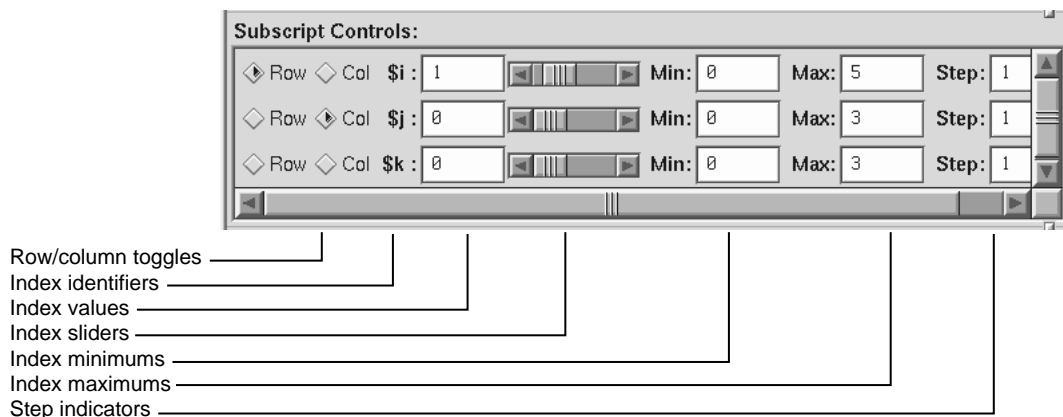
Figure 6-2 Subscript Controls Area in the **Array Browser**

The **Subscript Controls** area provides the following features:

- Row/column toggles: Controls whether an index variable represents rows or columns (or neither) in the spreadsheet area. You are not limited by the number of

dimensions of an array, but you can only view a two-dimensional orthogonal slice of the array at a time.

- Index identifiers: Indicates to which subscript the controls in the row refer.

- Index values: Shows the value of the subscript for the element currently in the focus cell. You can enter a different value if you wish.

- Index sliders: Lets you move the focus cell along the particular dimension.

- Index minimums: Identifies the beginning visible element in that particular dimension.

- Index maximums: Identifies the last visible element in that particular dimension. If you have an unspecified array, you can use this field to specify the last element in the vector to be displayed in the spreadsheet.

- Step indicators: Specifies the increment between adjacent elements in the dimension to be displayed. A value of 1 displays consecutive data. Specifying some n greater than 1 allows you to display every n element in a vector.

- Control area scroll bars: Allows you to expose hidden portions of the subscript control area if your window is not large enough for viewing all of the controls.

The spreadsheet area is where numeric data is displayed. It can show two dimensions at a time (indicated in the upper left corner of the matrix). The column indexes run along the top of the matrix and the row indexes are displayed along the left column. The spreadsheet area has scroll bars for viewing data elements not currently visible in the viewing area. Figure 6-3, page 63, shows a close-up of the spreadsheet area.
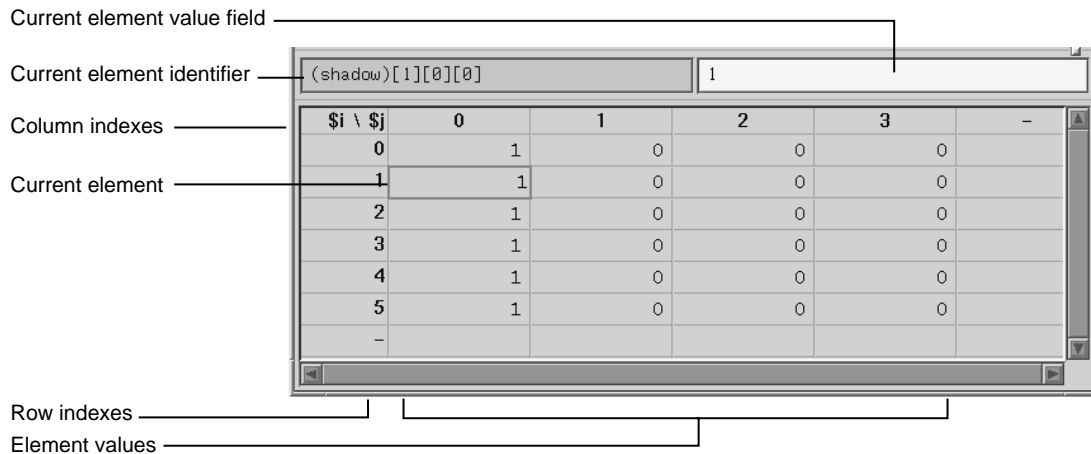
Figure 6-3 Array Browser Spreadsheet Area

The current element is highlighted by a colored rectangle in the spreadsheet area. Its corresponding expression is shown in the current element identifier field, and the value is shown in the current element value field.

The **Spreadsheet** menu allows you to change the appearance of data in the spreadsheet area. It provides these selections:

- **Column Width**: Allows you to specify the width of the spreadsheet cells in terms of characters. For instance, a value of 12 indicates that 12 characters, including punctuation and digits are viewable.

- **Wrapped Display**: Allows you to display a single dimension of an array wrapped around the entire spreadsheet area. The index value for an element is determined by adding the appropriate row index and column index values.

  Figure 6-4, page 64, shows an example of a wrapped array. There is only one index **$i**. The current cell is element 4 in the array (by adding 3 and +1).
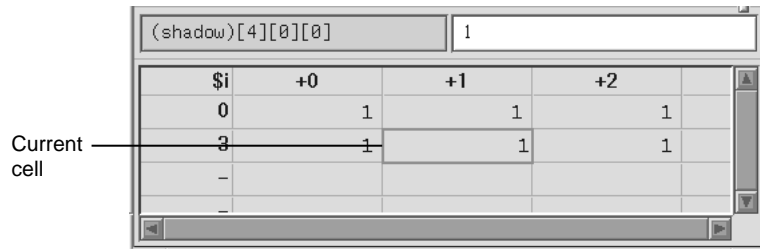
Figure 6-4 Example of Wrapped Array

- **Auto Fit Data**: If enabled, automatically resizes the spreadsheet cell to fit the maximum size of the data to be displayed. If this is enabled, then the **Column Width** is disabled. This option is on by default.

Format Menu

The **Format** menu displays a separate menu that you allows you to display the elements in the following formats:

- **Default** toggle: Toggles the default format.

- **Value** submenu: Contains the following display toggles for formatted values:

  Decimal
  Unsigned
  Octal
  Hex
  Float
  Exponential
  Char
  String
  Wide Character

- **Type**: Allows listing by data type.

- **Bit Size**: Allows listing by bit size.

The graphical display area presents array data in a three-dimensional graph in one of the following formats:

- Surface (polyhedron)

- Bar chart

- Points

- Multiple lines (array vectors)

Render Menu

The **Render** tear-off menu is available only if you are have an SGI workstation with Open Inventor installed on it.

You select the graphical display mode through the **Render** menu. The **Render** menu has the following options:

- **Surface**: Exhibits the data as a solid using the data values as vertices in a polyhedron.

- **Bar Chart**: Presents the data values as 3-D bar charts.

- **Points**: Simply plots the data values in 3-D space.

- **Multi Line**: Plots and connects the data values in each row.

- **None**: Allows you to display with no graphical display, in effect turning off graphical display mode.

Color Menu

The **Color** tear-off menu is available only if you are have an SGI workstation with Open Inventor installed on it.

If the **Color** menu is grayed out when the **Array Browser** window first opens, select the **Surface** option of the **Render** menu. The **Color** menu provides the following options:

- **Monotone Ramp**: Displays the data values in a single tone, with lower numbers being darker and higher values lighter in tone.

- **Hue Ramp**: Displays the data values in a spectrum of colors ranging from blue (lowest values) through green, yellow, orange, and red (highest values).

- **Exception**: Allows you to flag certain conditions by color, usually for the purpose of spotting bad data. When you select **Exception**, the controls shown in Figure 6-5, page 66 appear in the window.
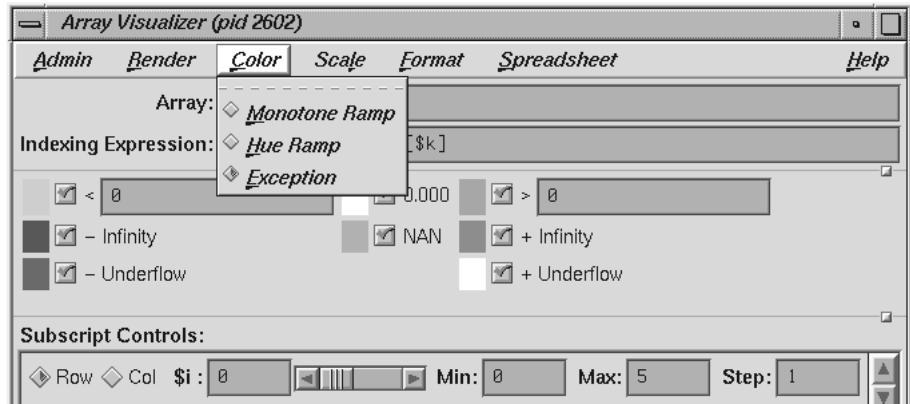
Figure 6-5 Color Exception Portion of **Array Browser** Window

Thus, you can highlight data values less than or greater than specified values,
values of plus or minus infinity, values of plus or minus underflow, zero values,
and NaN (not a number) values.

Surface rendering

Bar chart rendering

Point rendering

Multiple line rendering
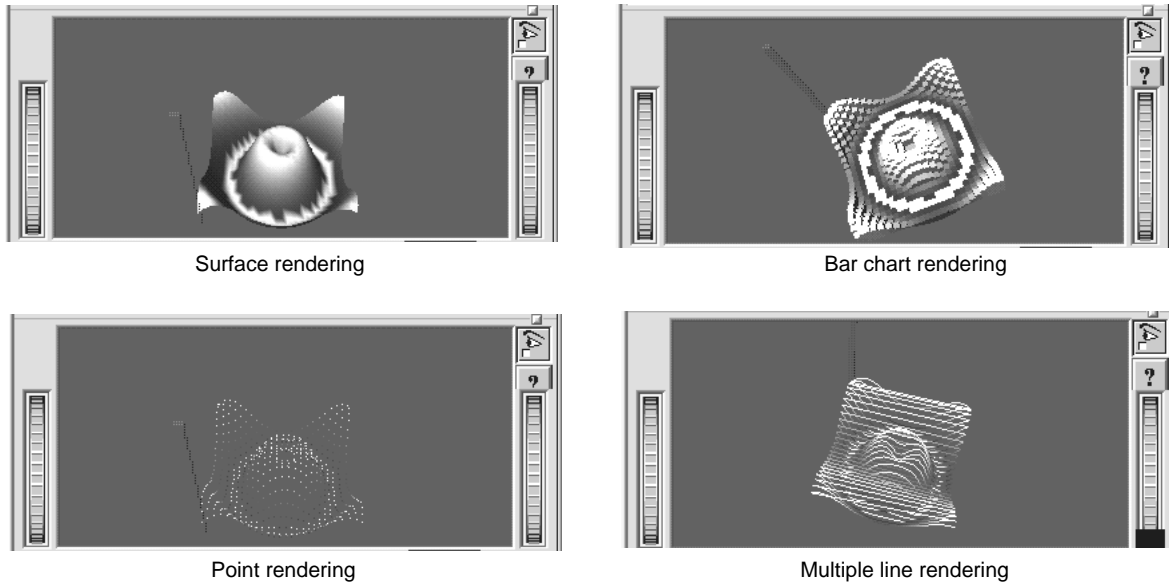
Figure 6-6 Array Browser Graphic Modes

Scale Menu

Note: The **Scale** tear-off menu is available only if you are have an SGI workstation with Open Inventor installed on it.

If the **Scale** menu is grayed out when the **Array Browser** window first opens, select the **Surface** option of the **Render** menu.

The **Scale** menu provides options for changing the ratio of the *z*-dimension, which represents the value of the element. The number on the left represents the value of the *x* and *y*-dimensions (which are always the same as each other). The number on the right is the *z*-dimension.

Manipulating the *z*-dimension affects the ease of spotting differences in values. If your data is scattered over a narrow range of values, you may wish to heighten the graph by selecting **10:1** as your scale; this exaggerates the values in the *z*-dimension. If your data is in a wide range, selecting **1:2** or **1:10** as the scale minimizes the differences, flattening the graph.

Examiner Viewer Controls

Note: The Examiner Viewer is available only if you are have an SGI workstation with Open Inventor installed on it.

The graphical display uses controls and menus from Examiner Viewer. Examiner Viewer is based on a camera metaphor and borrows terms from the film industry, such as *zoom* and *dolly*, in naming its controls. The graphical display area of the window is shown in Figure 6-7, page 69, with its main controls and menus. Note that the buttons on the upper right side of the graphical display area may not be visible if the area is too small; you can expose them by moving either the upper or lower sash to enlarge the display area. (The lines between the window panes include a small box to the right. Click-drag this box to change the size of the panes.)

Examiner Viewer provides these controls for viewing the graph. The right side buttons provide the following actions:

- **view mode**: Toggles between a view-only mode (closed eye) and manipulation mode (open eye).

  In view-only mode, the cursor appears as an arrow and the graph cannot be moved. Clicking on a portion of the graph selects the corresponding array element in the spreadsheet.

  In manipulation mode, the cursor appears as a hand and you can move the graph. Dragging the graph with the left mouse button down moves the graph in any direction as if it were in a trackball; a quick movement spins the graph. Dragging the graph with the left mouse button and the Ctrl key rolls (rotates) the graph in the plane of the screen. Dragging the graph with the middle mouse button moves it without changing the viewing angle.

  If you drag the graph with both the left and middle mouse buttons down, the graph appears to move into or out of the window (this is the same as the **dolly thumbwheel**, which is described in this section).

View mode

Help

Home

Set home

View all

Seek

Dolly control

Rotx Roty Zoom 45.0 Dolly

x rotation control

y rotation control

Zoom control and readout

Functions Tear-off
Help
Home
Set Home
View All
Seek
Copy View
Paste View

Examiner Viewer
Functions
Draw Style
Viewing
Decoration
Headlight
Preferences...

Draw Style
as is
hidden line
no texture
low resolution
wireframe
points
bounding box (no depth)
move same as still
move no texture
move low res
move wireframe
move low res wireframe (no depth)
move points
move low res points (no depth)
move bounding box (no depth)
single buffer
double buffer
interactive buffer

Figure 6-7 Examiner Viewer with Controls and Menus

- **help**: Runs a special help system containing Inventor Viewer information.

- **home**: Repositions the graph in its original viewing position.

- **set home**: Changes the home (original viewing) position for subsequent use of the **home** button.

- **view all**: Repositions the display area so that the entire graph is visible.

- **seek**: Provides a special cursor that allows you to reposition the graph in the center of the display area or allows you to center the view on a point you select with the cursor. See **Seek to point <or object>** in the Preferences dialog box.

The following controls let you move the graphic display:

- **x rotation thumbwheel**: Rotates the graph around its x-axis.

- **y rotation thumbwheel**: Rotates the graph around its y-axis.

- **dolly thumbwheel**: Changes the size of the graph and adjusts the angles to maintain perspective. The dolly control simulates moving the viewing camera back and forth with respect to the graph.

Examiner Viewer Menu

You access the **Examiner Viewer** menu by holding down the right mouse button in the graphical display area. The **Examiner Viewer** menu provides the following options (see Figure 6-7, page 69 for illustration):

- **Functions**: Displays a submenu with the selections **Help**, **Home**, **Set Home**, **View All**, and **Seek**, which are the same as the right mouse button controls described in the previous section, and the **Copy View** and **Paste View** selections. These operate like standard copy and paste editing commands, enabling you to transfer graphs.

- **Draw Style**: Displays a submenu that controls how the graph is displayed. The top group of options, from **as is** to **bounding box (no depth)** control how the graph is displayed when it is static. These override any **Render** menu selections.

  The middle (**move...**)group of options control how the graph is displayed while in motion.

  The last three options, **single**, **double**, and **interactive**, refer to buffering techniques used in moving the graph. These affect the smoothness of the movement.

- **Viewing**: The same as the **view mode** button described in the previous section. When it is off, you can select points from the graph to display in the spreadsheet but cannot move the graph. When on, it allows you to manipulate the graph.

- **Decoration**: Displays the right side buttons when it is on and hides them when it is off.

- **Headlight**: Controls the shadow effect on the graph. When it is on, the light appears to come from the camera.

- **Preferences**: Causes the **Examiner Viewer Preferences Sheet** dialog to display.
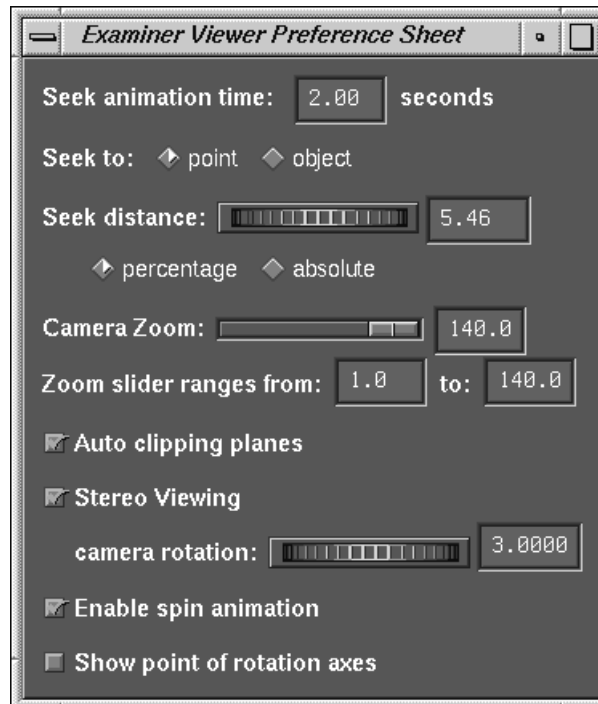
Figure 6-8 Examiner Viewer Preference Sheet Dialog

The **Examiner Viewer Preference Sheet** dialog provides the following options:

- **Seek animation time**: Allows you to specify the time it takes for the graph to be repositioned after you change the seek point. Set to 0 for instant seek. See also **Seek to point <or object>**.

- **Seek to point <or object>**: Seek to point uses the picked point and surface normal to align the camera.

  Seek to object uses only the object center to align the camera.

- **Seek distance** wheel: This wheel controls how close to the camera the object appears. This distance can be either an absolute distance or a percentage of the distance to the picked point.

- **Camera Zoom** slider: This slider allows you to set (in degrees) the camera height angle (only perspective camera). If you wish, you may set this value in the field immediately to the right of the slider.

- **Zoom slider ranges from**: Range is **1.0** to **140.0** by default, but you can reset this if you wish.

- **Auto clipping planes**: Centers the graph in your view if enabled. If disabled, it allows you to move the graph out of view at either end of the z-axis. This is useful if you wish to focus on data above or below a set value.

- **Stereo Viewing**: This may be turned on to see the scene in stereo (special glasses required). The offset between the left and right eye can be specified using the **camera rotation** thumbwheel.

- **Enable spin automation**: When selected, you can cause the camera to continue spinning. To animate, left-click and drag in the direction of your desired spin, then release while spinning. The graph spins as if by a trackball. To stop, left-click anywhere.

- **Show point of rotation axes**: Displays a set of three axes. You can move the graph around the x and y axes using the thumbwheel controls described in the previous section. When this option is on, you can set the size of the axes in pixels.

Call Stack Window

The **Call Stack** window displays call stack entries when a process has stopped.

The source display in the Main View window has two special annotations that are synchronized with the **Call Stack**:

- The location of the current program state is indicated by a large arrow representing the PC. The source line to which the arrow points is highlighted (usually in green).

- The location of the call to the item selected in the **Call Stack** window is indicated by a small arrow representing the current context. The source line becomes highlighted (usually in blue-green).

The **Call Stack** contains its own menu bar, which contains the **Admin**, **Config**, **Display**, and **Help** menus. The **Admin** menu is the same as that described in "Admin Menu, Task View", page 29. The **Help** menu is the same as that described in "Help Menu", page 17. The other menus are described in the following list:

- The **Config** menu contains the **Preferences** option, which launches the **Call Stack Preferences** dialog that allows you the option of setting the maximum depth of the **Call Stack**.

- The **Display** menu contains the following toggles which change what is displayed in each entry of the **Call Stack** in addition to the subroutine/function name:

  - **Arg Values**: Allows you to display argument values. Default is on.

  - **Arg Names**: Allows you to display argument names. Default is on.

  - **Arg Types**: Allows you to display argument types. Default is off.

  - **Location**: Allows you to display function location. Default is on.

  - **PC**: Allows you to display the program counter (PC). Default is off.

Data View Window

The Data View opens a window that allows you to display variables and their values in the source code.

Expression View Window

Expression View displays a collection of expressions that are evaluated each time the process stops or the context changes.

In addition to the items on the menu bar, the window has two pop-up menus: the **Language** menu and the **Format** menu. The **Admin** menu is the same as that described in "Admin Menu, Task View", page 29. The **Help** menu is the same as that

described in "Help Menu", page 17. The other menus are described in the following list:

- The **Config** menu contains the following options:

  – **Load Expressions**: Launches the **Load Expressions** dialog box that allows you to choose source file from which to load your expressions.

  – **Save Expressions**: Launches the **Save Expressions** dialog box that allows you to choose a file to which you can save your expressions.

- The **Display** menu contains the following options:

  – **Format All**, which allows you to change the default values. See "Format Pop-up Menu", page 74 for a description of format options.

  – **Clear All** option which clears all fields in the view.

Language Pop-up Menu

The **Language** pop-up menu contains three buttons that allow you to select one of three languages for evaluation: C, C++, or Fortran. The **Language** pop-up is invoked by holding down the right mouse button while the cursor is in the **Expression** column.

Format Pop-up Menu

The **Format** pop-up menu is displayed by holding down the right mouse button in the **Result** column.
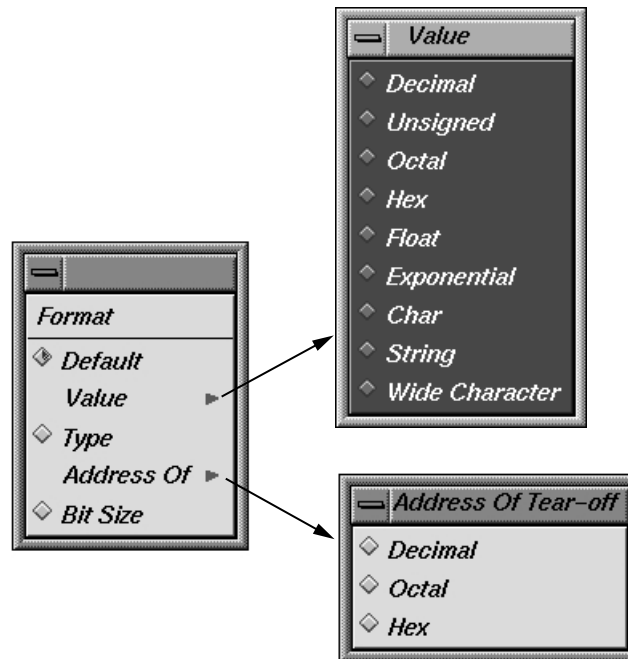
Figure 6-9 Expression View Format Popup with Submenus

The **Format** popup contains the following options:

- **Default**: Sets the format to the default values.

- **Value**: Displays a submenu from which you can select a value of **Decimal**, **Unsigned**, **Octal**, **Hex**, **Exponential**, **Float**, **Char**, **String**, or **Wide Character** type.

- **Type**: Displays a submenu from which you can select a type of **Decimal**, **Octal**, or **Hex**.

- **Bit size**: Sets the format to **Bit Size**.

File Browser Window

The **File Browser** window displays a list of source files used by the current executable. Double-click on a file in the list to load it into the source display area in

the Main View or **Source View** windows. Some files may be listed due to subroutines/functions being resolved from system libraries. If you select such a file, you may get the following message:

```
Unable to find file <xxx.c>
```

This is because the source for system library routines may not be stored on your system.

The **Search** field allows you to find files in the list.
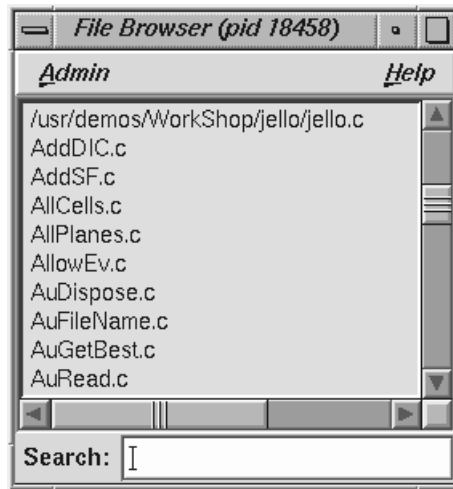
Figure 6-10 **File Browser** Window

The **File Browser** contains an **Admin** menu (described in "Admin Menu, Task View", page 29) and a **Help** menu (described in "Help Menu", page 17). In addition, the browser has a **Search** field.

Data Explorer Window

The **Data Explorer** window allows you to examine variables, data structures, Fortran common blocks, and the relationships of the data within them. It displays complex data structures as separate graphical objects, using arrows to indicate relationships. The variable list on the right is a list of local variable available to be displayed for the

current stack frame of the current process/thread. The list can display as one of the following:

A sorted list of local variables:
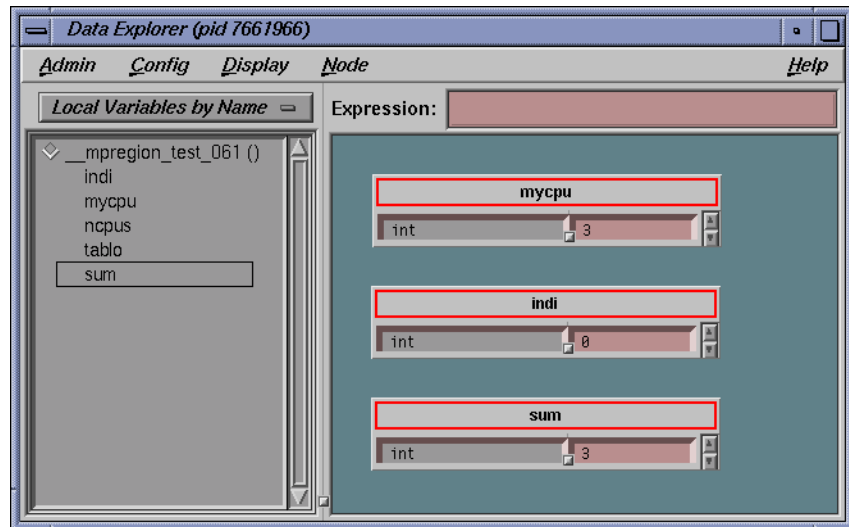
Figure 6-11 Data Explorer: reference by name

Names grouped by type:
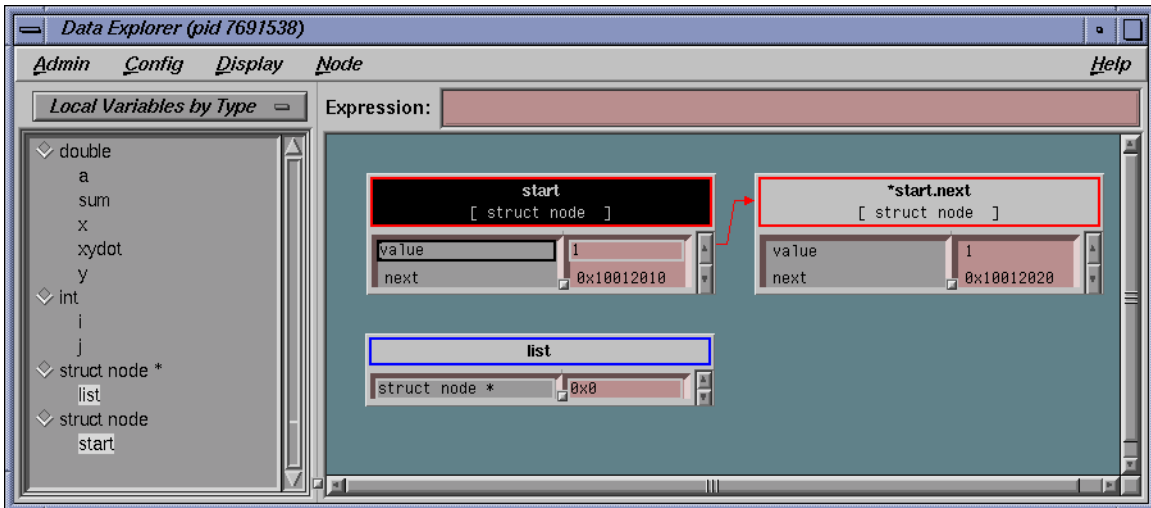
Figure 6-12 Data Explorer: names grouped by type

Common blocks, if the current frame is part of a Fortran application:
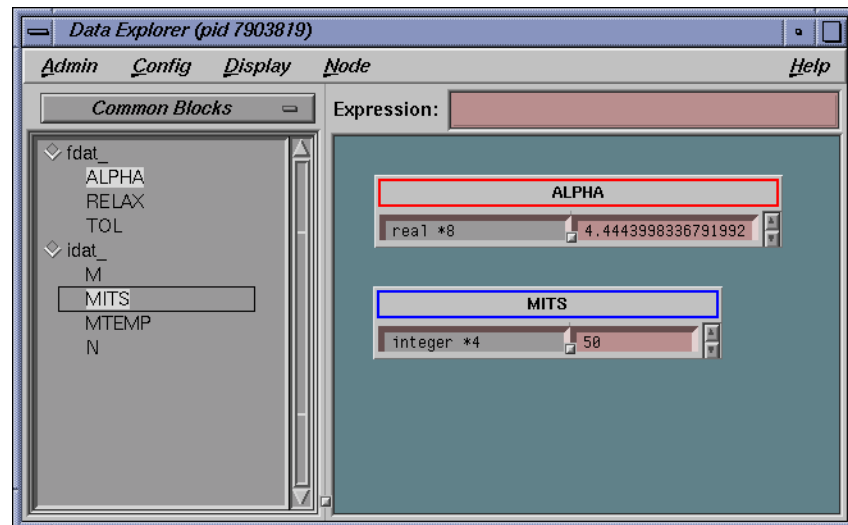
Figure 6-13 Data Explorer: common blocks

A data name is entered in the **Expression** field or selected by a left or right mouse click on a name in the variable list to the left of the object display area. It then appears as an object or set of objects in the display area in the lower portion of the window. The variable list represents local variables for the current process frame. Other variables that are not of local scope (for example, globals) can be entered into the **Expression** field. Each data item has a header identifying the data, color coded by data type. Below the header are two columns: the left displays the field name and the right displays the field's value. If a displayed data item exceeds the size of the **Data Explorer** window, scroll bars appear.

The **Admin** menu contains options for selecting the active structure, cloning structures, saving them as text, and closing the **Data Explorer** window.

The other menus in this window contain options that allow you to change the way data is displayed. The following data-display menus are available:

- **Config**: Provides options for saving and reusing type-specific formats and expressions. You can also set preferences regarding how objects of a given type are to be displayed.

- **Display**: Provides display options for all objects in the display area.

- **Node**: Provides options for selected objects in the display area only.

The **Format** menu appears in the structure header display area. It allows you to change or reformat a specific value in the result column. To access this pop-up menu, hold down the right mouse button while the cursor is over the result column.

Using the Data Explorer Overview Window to Navigate

WorkShop provides the **Data Explorer Overview** window (from the **Show Overview** option in the **Display** menu) as another way to navigate around the display.

This window is a reduced-scale view of the requested structures. The structures are represented by solid rectangles color-coded by data type. The relative position of the currently visible area is represented by a transparent rectangle. This rectangle can be dragged (using the left mouse button) to change the display of the **Data Explorer**. Clicking the left mouse button in an area of this window repositions the currently visible area.

Entering Expressions

The **Data Explorer** accepts any valid expression. If the result type is simple, a structure displays showing the type and value. If the result type is a pointer, the pointer value and type are displayed (if the Data Explorer preference item is set, it is automatically de-referenced until a non-pointer type is reached).

Figure 6-14 Dereference indicator on **Data Explorer Preference** menu

If the result type is a structure or union, an object is displayed showing the structures' fields and their values. After the expression is entered, the **Expression** field clears. The **Data Explorer** can display unrelated structures at the same time; you simply enter new structures by using the **Expression** field.

The **Expression** field is also used to enter strings used in searches.

Working in the Data Explorer Display Area

Within the display area, you select objects by clicking in the node headers. Shift-clicks add the selected object to the current selection. You can drag selected objects using the middle mouse button.

Clicking the right button while the cursor is in the right column of an object displays the **Format** menu, which is used to change the display. You can set a default format or request that results be displayed by value, type, address, or size in bits.

Holding down the right button in the header of an object brings up the **Node** pop-up menu, which is the same as the **Node** menu in the menu bar. It is used to change the way selected objects are displayed. When you left-click in the header of an object, it turns on the resizer, which allows you to change the size of the object. You see a small square (handle) at the upper-right and lower-left corners. Left-click-drag the handle to resize the object. Middle-click–drag the handle to move the object.

Arrows show relationships among the displayed structures. If a member field is not visible in a structure, its arrow tail is displayed at the top or bottom of the scrolling area for fields. Otherwise, its tail is adjacent to its field.

Double-clicking a value field (right column) for a pointer changes the display so that the data structure it points to is displayed.

Data Explorer Display Menu

The **Display** menu controls the way structures appear in the display area. The **Display** menu provides the following options:

- **Display**: Determines contents of the display. The **Display** option has the following two options:

  - **Expression** — Displays the structure of the expression entered in the **Expression** field.

  - **Selection** — Displays the structure based on the text you have selected in the source code pane in the Main View window.

- **Arrange**: Rearranges the currently selected nodes. **Arrange** has the following two options: (See Figure 6-15, page 82.)

  - **Tree** — Arranges nodes into a tree-type formation, that is, the hierarchy descends from left to right and child structures are shown as branches to the right of the parent.

– **Linked List** — Arranges nodes into a linked list formation, that is, horizontally.
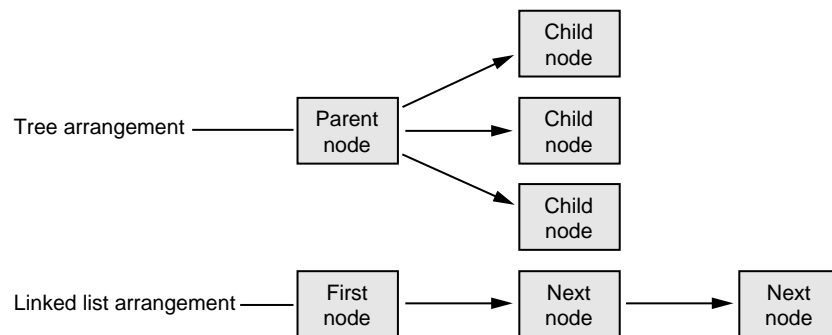
Figure 6-15 Tree and Linked List Arrangements of Structures

• **Search**: Allows you to select structures in the display area that contain the *string* specified in the **Expression** field. **Search** has the following four options:

– **Name** — Selects structures whose names contain the specified string.

– **Type** — Selects structures whose types contain the specified string.

– **Field Name** — Selects structures that have a field whose name contains the specified string.

– **Value** — Selects structures that have a field value containing the specified string.

– **Update**: Explicitly updates the displayed structures. This happens automatically in the current **Data Explorer** when the process stops. This can be used in an inactive **Data Explorer** to update it. It can also be used to update the display after changes have been made in other Debugger views.

– **Show Overview**: Brings up the **Data Explorer Overview** window.

– **Clear All**: Clears all structures from the display area.

Node Menu

The **Node** menu gives you options that apply to currently selected objects.

The following menu items are available:

- **State**: Controls the display of nodes. There are three options:

  - **Iconic** — Displays the node header only.

  - **Normal** — Uses the default chart display but hides those fields selected to be invisible.

  - **Detail** — Uses the default chart display and shows all fields.

- **Geometry**: Manages graphical objects in the display area. There are four options:

  - **Minimize** — Sets the vertical size of an object to the default minimum number of fields. The initial default is four fields but can be changed through either the **Formatting** selection from the **Node** menu or the **Preferences** selection from the **Config** menu.

  - **Maximize** — Displays the object as large vertically as necessary to fit all of the fields.

  - **Raise** — Raises the selected object(s) to the top of the display.

  - **Lower** — Lowers the selected object(s) to the bottom of the display.

- **Select**: Allows you to select objects in various ways. There are six options:

  - **Parents** — Selects all objects that have pointers pointing to a selected object.

  - **Children** — Selects all objects pointed to by any fields in a selected object.

  - **Ancestors** — Selects all objects pointed to a selected object or pointing to an object that has a descendant pointing to a selected object.

  - **Descendant** — Selects all objects pointed to by any fields in a selected object or pointed to by any children of a selected object.

  - **Family** — Selects all ancestors and descendants of a selected object.

  - **All** — Selects all objects.

- **Other Views**: Brings up other view options, such as **Expression View**, **Data Explorer**, etc.

- **Formatting**: Brings up the type formatting dialog for this type.

- **Dereference Ptrs**: Dereferences any pointers in selected objects.

- **Pattern Layout**: Displays selected structures that are connected by pointers to position related structures in the same way.

- **Remove**: Removes selected object from the display.

Formatting Fields

Each field in a data structure has certain display characteristics. These can be specified for all objects in the **Data Explorer Preferences** dialog box or for type-specific objects only in the **Data Explorer Type Formatting** dialog box. To display the **Data Explorer Preferences** dialog box, select **Preferences** from the **Config** menu.

The dialog has the following fields:

- **Default Structure Field Count**: Sets the number of fields to be displayed initially.

- **Default Structure Width**: The width in pixels of the object.

- **Default Iconic Width**; The width in pixels of the object when it is in iconic form.

- **Automatic Dereference Limit**: Limits the number of structures that are automatically dereferenced.

- **Dereference Ptrs By Default**: Toggles automatic dereferencing on and off.

- **Use dbx Style Evaluation**: Toggles to use a dbx evaluation.

To bring up the **Data Explorer Type Formatting** dialog box, select the set of structures under consideration and select **Node Formatting** from the **Node** menu.

The dialog box has the following fields:

- **Type Name**: Displays the current data type.

- **Default Field Count**: Lists number of fields to be displayed initially for objects of that type.

- **Default Structure Width**: Displays the width in pixels of the object.

- **Default Iconic Width**: Displays the width in pixels of the object when it is in iconic form.

- **Default State**: Brings up a pop-up menu that allows you to specify whether structures are first displayed as icons (**Iconic**), with the minimum number of fields displayed (**Normal**) or with all fields displayed (**Detail**).

- **Type Color**: Provides a submenu for color coding. It allows you to select a color for the header and overview rectangles for objects of a given type.

For structure and union types, the list box shows all the fields with their types. For each field, you can change the result format to one of the following types:

- Default

- Decimal

- Unsigned

- Octal

- Hex

- Float

- Exponential

- Char

- String

- Wide Character

- Type

- Dec addr

- Oct addr

- Hex addr

- Bit Size

You can also specify whether a field is visible in normal state, and if it is a pointer field, whether it should be automatically dereferenced.

After you specify the format for this type, you can apply it to any combination of the following through the toggle buttons in the bottom left portion of the window:

- Selected instances

- All existing instances

- Any future instances of this type

Variable Browser Window

The **Variable Browser** window allows you to view and change the values of local variables and arguments at a specific point in a process. (Global variables can be viewed or changed using **Expression View** or the **Evaluate Expression** selection from the **Data** menu for one-shot evaluations.) In addition to providing values, the **Variable Browser** is useful for getting a quick list of the local variables in a scope without having to search for their names..

Typically, you inspect variable values at the following points:

- At a breakpoint

- At a frame in a call stack

- As you step through a process

A useful technique is to set a trap at the entry to a function and inspect the values of the variables there. Some variables may be in an uninitialized state at that point. You can then step through the function and make sure that no uninitialized variables are used inadvertently.

Note: In programs compiled by using the `-n32 -g` compiler options, you cannot click on or print an unused variable as you could when using the `-o32 -g` options.

Entering Variable Values

The **Variable Browser** allows you to change the values of variables in the window. You simply enter the new value in the result column and press `Enter`. Thus, you can force new values into the process and see their effect.

Changing Variable Column Widths

By using the sash between the columns, you can adjust the relative widths of the **Variable** and **Result** columns. For example, you may wish to adjust for short variable names and long result values.

Viewing Variable Changes

The Debugger views that are involved with variables (that is, the **Variable Browser** and **Expression View**) have indicators that show when the variable has changed since the last breakpoint. If you click the indicator, you can view the previous value.

Machine-level Debugging Windows

The Debugger offers three views useful in debugging at the machine level: the **Disassembly View**, **Register View**, and **Memory View**.

The Disassembly View Window

The **Disassembly View** window allows you to look at machine-level code rather than source-level code. A typical **Disassembly View** window is brought up by selecting **Disassembly View** from the Main View window menu bar.

Similarities with Main View Window

At the top of the window are the same process control buttons as those in the Main View window. They behave the same way except for **Step Into** and **Step Over**, which do machine-level instruction stepping instead of source-level. Remember that you can select the number of steps by holding down the right mouse button over the **Step Into** and **Step Over** buttons.

The **Admin**, **Display**, and **Traps** items on the menu bar basically work same as their counterparts in the Main View window.

The **Disassemble** menu is discussed in "The Disassemble Menu", page 89. The **Preferences** selection from the **Config** menu is discussed in "The Config Menu Preferences Dialog", page 91.

You can set traps either by using the **Traps** menu or by clicking in the annotation column to the left of the source display area that contains the disassembled code.

The Disassemble Menu

The **Disassemble** menu allows you to display disassembled code. It contains the following items:

- **Address**: Allows you to disassemble a specified number of lines, starting from a specified source line address (see Figure 7-1).
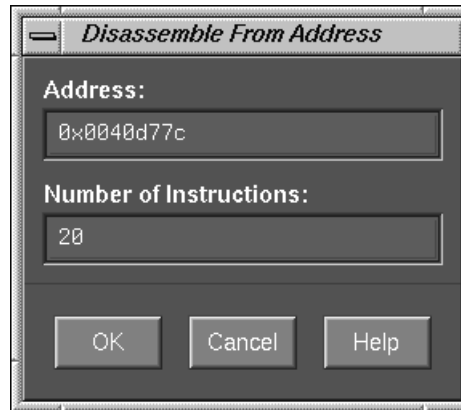
Figure 7-1 The **Disassemble From Address** Dialog

- **Function**: Allows you to disassemble a specified number of lines, starting from the beginning address of a specified function name.

- **File**: Allows you to disassemble a specified number of lines, starting from the address corresponding to a specified line number in a specified file (refer to Figure 7-2, page 91). If you have a current selection in the Main View window or the **Source View** window, its file and cursor position are used as the default file name and line number, respectively.

Figure 7-2 The **Disassemble File** Dialog

The Config Menu Preferences Dialog

Selecting **Preferences** from the **Config** menu brings up the **Disassembly View Preferences** dialog box so that you can select the display preferences you desire.

The dialog box provides you with the following options:

- **Number of instructions to disassemble**: Controls the default number of disassembly lines shown when the process stops. This number appears in the dialog boxes selected from the **Disassemble** menu. The default is all instructions, indicating that the entire function will be disassembled.

- **Minimum lines around current instruction**: Controls the display of the disassembled code, enabling you to view at least the specified number of instructions before and after the current instruction.

- **Register name display format**: Controls how register names are displayed. The available modes are **Hardware**, **Compiler**, and **Assembler**.

- **Show embedded source annotation**: When **ON**, displays source and disassembly statements interleaved. When **OFF**, displays disassembly statements only.

- **Show source file and line number**: Displays the filename and file position along with each machine instruction.

- **Show function name and line number**; Displays the function name and file position along with each machine instruction.

- **Show machine address**: Displays the memory address of each machine instruction.

- **Show instruction value**: Displays the instruction word along with each machine instruction.

- **Show jal targets numerically**: Controls whether the target address of a `jal` instruction is displayed as a hex address or symbolic label.

- **Show supplemental address info (pixie, cord, original)**: Displays additional address information. This may be used to set address breakpoints in the Main View command window for corded or pixified code.

The Register View Window

Register View window allows you to examine and modify register values. You bring it up by selecting **Register View** from the **Views** menu in the Main View window. Figure 7-3, page 93, shows a typical **Register View** window that has been resized to show all available registers.

The **Register View** window displays each register with its current value. A question mark (?) displayed immediately before a register value signifies that the value is suspect; it may not be valid for the current frame. This can occur if a register is not saved across a function call. A colored marker indicates that a register value has changed since the last time the process stopped.

Current register value field   Modify button
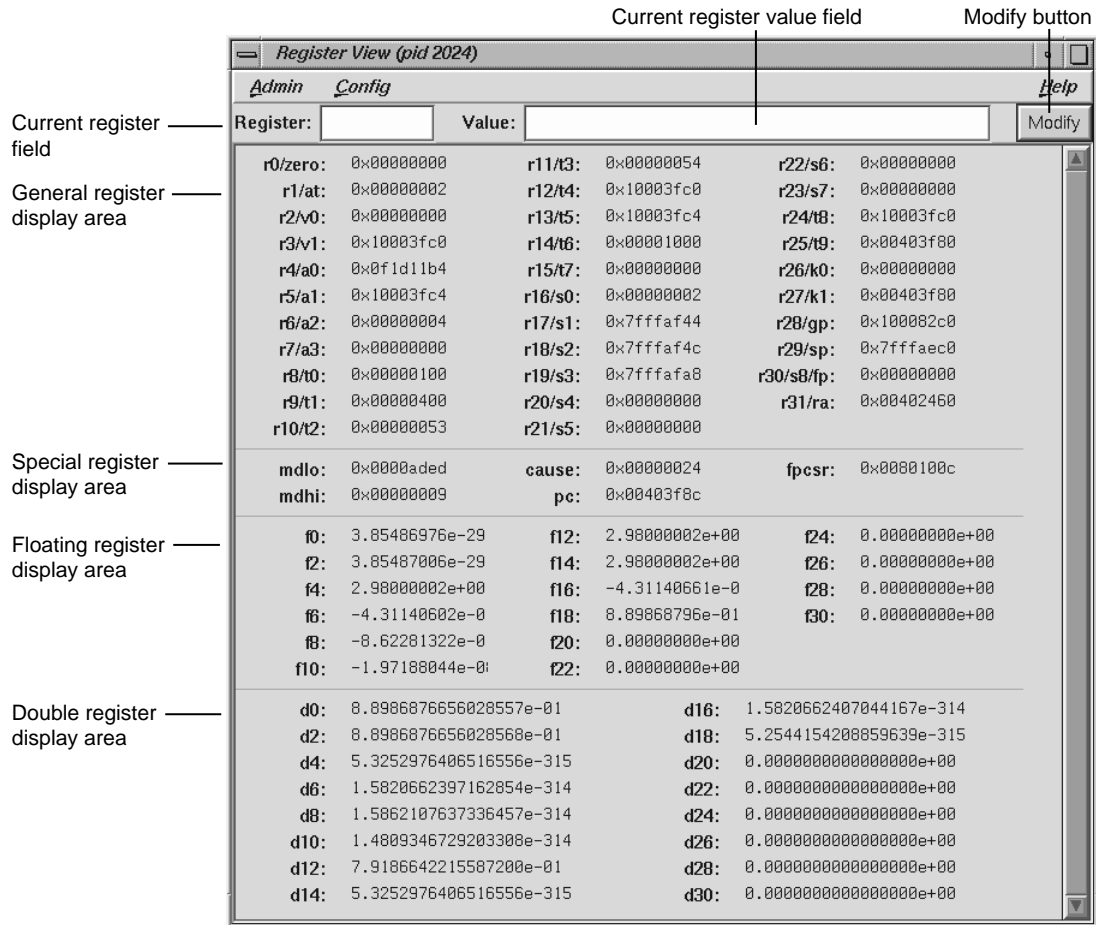
Figure 7-3 The **Register View** Window

The major features of the **Register View** window are the following:

- Current register field: Identifies the currently selected register. You can switch to a different register by entering its name (either by hardware name or by alias) in this field and pressing `Enter`. You can also switch registers by clicking on the new register in the display area.

- Current register value field: Shows the contents of the selected register. You can assign a new value to a register by entering either a literal or an expression into

the **Value** field. You then click on the **Modify** button to change the value or press `Enter`.

- Register display area: Shows the registers organized into four groups: general, special, floating, and double. Note that the general registers are identified by both their hardware and software names. For systems with 32–bit processors (O2s, for example), double precision registers represent a pair of floating-point registers. For systems with 64–bit processors (Origin2000s, for example), float registers are not displayed at all. Floating-point calculations are done in double precision registers.

Note: The special registers **p0**, `p1`, and **p2** do not appear in Figure 7-3, page 93. These are used for instrumentation and appear only when instrumentation has taken place.

Changing the Register View Display

The **Preferences** selection in the **Config** menu allows you to change the **Register View** display. It brings up the **Register View Preferences** dialog box (see Figure 7-4, page 95).

The register display toggle buttons let you specify which types of registers are to be displayed by default.
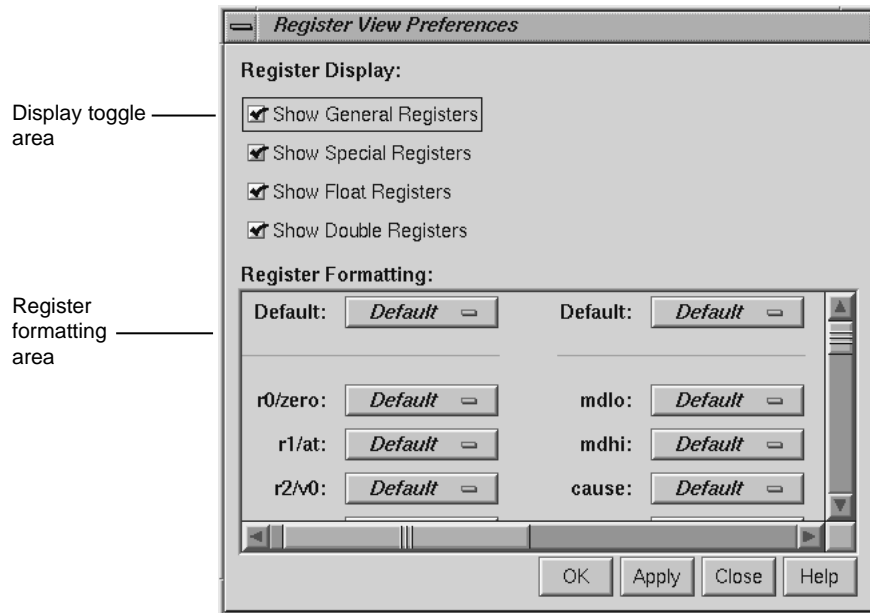
Figure 7-4 The **Register View Preferences** Dialog

The register formatting area allows you to select formats for any of the registers.

The default fields in the top row let you change defaults for the four major types, which are set as follows:

- General registers — hexadecimal

- Special registers — hexadecimal

- Float registers — floating point

- Double registers — floating point

The rows in the register formatting area let you change the modes for the individual registers.

The Memory View Window

The **Memory View** window allows you to examine and modify memory. A typical **Memory View** window appears in Figure 7-5, page 96.
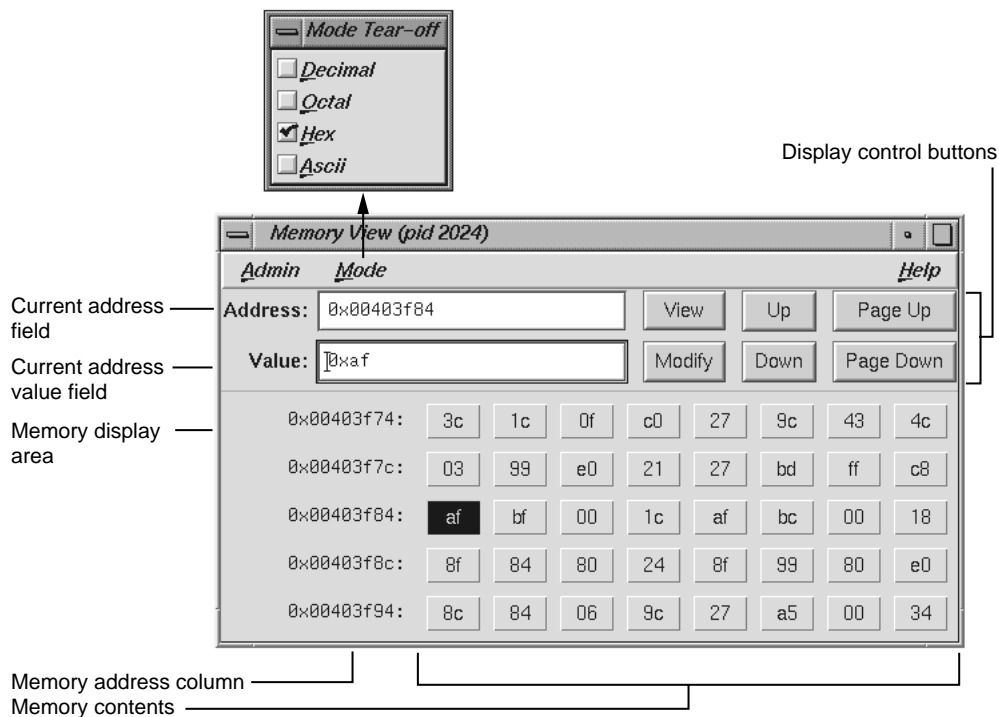
Figure 7-5 The **Memory View** Window with the **Mode** Submenu Displayed

Viewing a Portion of Memory

To view a portion of memory, enter the beginning memory location in the **Address** field. You can enter the literal value or an expression that evaluates to an integer address. These address specifications must be in the language of the current process as indicated by the call stack frame. The syntax of this expression depends upon the language of the program. For example, you can enter **0x7fff4000+4** as the memory address when stopped in a C function or enter **$7fff4000+4** as the equivalent for a Fortran routine. Press Enter while the cursor is in the field or click the **View** button to display the contents of that location and the subsequent locations in the display

area. This also displays the contents of the first address in the **Value** field where it can be modified.

The memory display area shows the contents of individual byte addresses. The column at the left of the display shows the first address in the row. The contents at that address are shown immediately to its right, followed by the contents of the next seven byte locations. If you enlarge the **Memory View** window, you can see additional rows of memory.

Changing the Contents of a Memory Location

To change the contents of a memory location, you select the address to be changed, either by direct entry or by clicking on the byte value in the display area. You can enter a single value or a sequence of hex byte values separated by spaces (for example, `00 3a 07 b2`) in the **Value** field. You can also enter a quoted string to change a consecutive range of values to the ASCII values of that string. Pressing `Enter` while the cursor is in the **Value** field or clicking the **Modify** button substitutes the new value(s) starting at the specified location.

Changing the Memory Display Format

The **Mode** menu allows you to change the format of the value field and byte locations to either decimal, octal, hex, or ASCII.

Moving around the Memory View Display Area

The four control buttons at the upper right of the window help you move around the display area. These buttons are:

- **Up**: Moves displayed bytes up a single row.

- **Down**: Moves displayed bytes down a single row.

- **Page Up**P Moves displayed bytes upward by as many rows as are currently displayed.

- **Page Down**: Moves displayed bytes downward by as many rows as are currently displayed.

Fix+Continue Windows

The Fix+Continue utility interacts with several Debugger windows. The Main View and **Source View** windows access the Fix+Continue utility from the menu bars. The standard output results of running redefined code are displayed in the **Execution View** window (refer to "Changes to Debugger Views", page 105).

Special line numbers (in decimal notation) applied to redefined functions appear in several views.

Note: Fix+Continue functionality within the debugger is limited to C++ programs compiled with the −o32 compiler option.

The following windows, devoted entirely to Fix+Continue functions, can be brought up by selecting **Fix+Continue > View** from the Main View window menu bar:

- **Status Window**

- **Message Window**

- **Build Environment Window**

This chapter describes Fix+Continue menu selections and windows.

The **Fix+Continue** menu is available from the Main View menu bar. The menu selections operate on the selected code or the file shown in the **Source View** window. The **Fix+Continue** menu is also available from **Source View** window and from the **Fix+Continue Status** window.

Fix+Continue Status Window

The **Fix+Continue Status** window (see Figure 8-1, page 100) provides you with a summary of the modifications you have made during your session. It also allows you quick access to your modifications and somewhat expanded menu options.
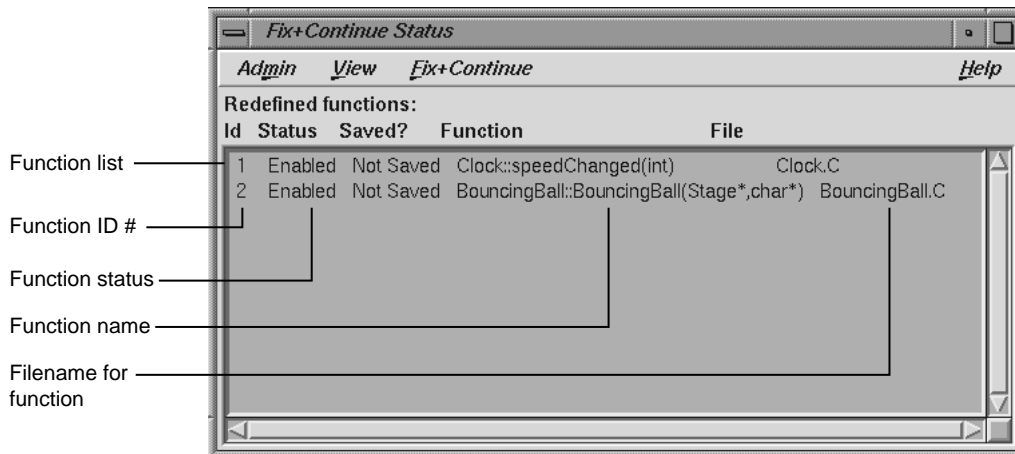
Figure 8-1 Fix+Continue Status Window

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

The following menus and options are available:

- The **Admin** menu contains an option for closing the window.

- The **View** menu contains options for sorting information in the window and displaying file names.

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

  - **Show Long Filenames**: Toggles among absolute (long) path names or base names.

- The **Fix+Continue** menu available from the **Fix+Continue Status** window is somewhat different from that available through the Debugger Main View. It contains a number of options and submenus that are described below. These options and submenus are active on the item you select in the source pane of the Main View window. Before using this menu, you should select an item by clicking on it. The following options and submenus are available:

– **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 code and loads it for execution. You can execute the modified code by clicking on the **Run** or **Continue** command buttons in the Main View window.

– **Update All Files**: Launches the **Save File+Fixes As** dialog that allows you to update the current session while saving all the modifications to the appropriate files.

– **Show Difference** submenu: 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.

– **Enable** submenu: Allows you to enable the changes in your modified code in one of the following ways:

   • **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: Has the same menu choices as the **Enable** submenu, but disables rather than enables.

– **Save** submenu: Allows you to save your code changes to a file. You can save the changes in one of the following ways:

   • **Function** — Launches the File dialog, allowing you to save only the current function to a file.

   • **File** — Launches the **Save File+Fixes As** pop-up window allowing you to save the entire file that contains the current function.

– **Delete** submenu: Has the same menu choices as the **Save** submenu, but deletes rather than saves.

– **Show** submenu: Allows you to launch any of the following windows:

- **Message Window** — Launches a **Fix+Continue Error Messages** window for the selected item. See "Fix+Continue Message Window", page 102, for more details.

- **Build Env for File** — Launches a **Fix+Continue Build Environment** window for the file shown in the **Source View** window. See "Fix+Continue Build Environment Window", page 103, for more details on the **Fix+Continue Build Environment** window.

- **Default Build Env** — Launches the **Fix+Continue 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", page 103, for details on the **Fix+Continue Build Environment** window.

Fix+Continue Message Window

The **Fix+Continue Message** window contains a list of errors and other system messages that pertain to your source modifications, parses, and attempts to run your modified source.

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

- **Clear**: Clears all the parsing errors and warnings.

- **Next**: Puts a check mark on the next unchecked 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. The **Next** option does not work 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.

Admin Menu

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

- **Clear All**: Clears all messages in the window.

- **Close**: Closes the window.

View Menu

The **View** menu allows you to set any of the following 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 8-2, page 104). The **Fix+Continue 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.
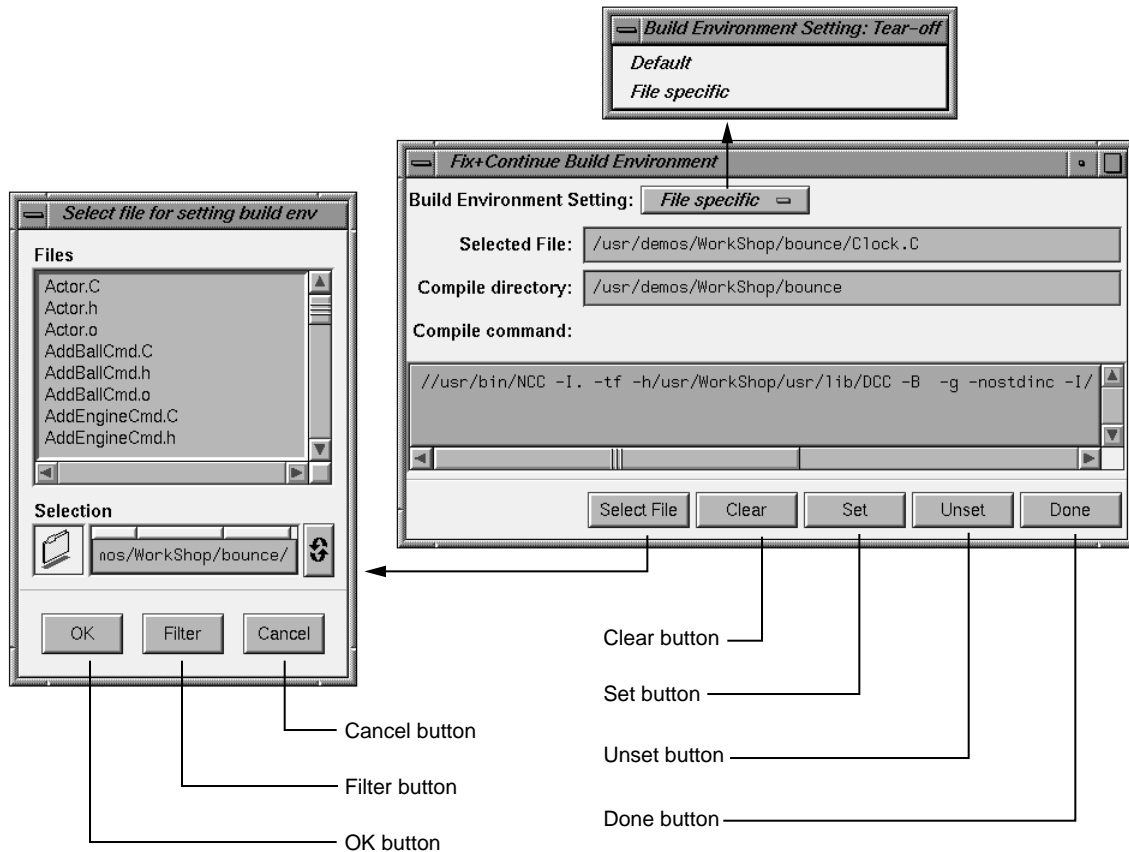
Figure 8-2 Fix+Continue Build Environment Window

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

The window allows you to select your build environment setting through the **Build Environment Setting** toggle that contains the following two options:

- **Default**: Sets the build environment to default that is displayed in the 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.

The **Fix+Continue 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.

Changes to Debugger Views

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

Main View

All Fix+Continue actions are available through the **Fix+Continue** menu on the Main View window. See Figure 8-3, page 106, for details.
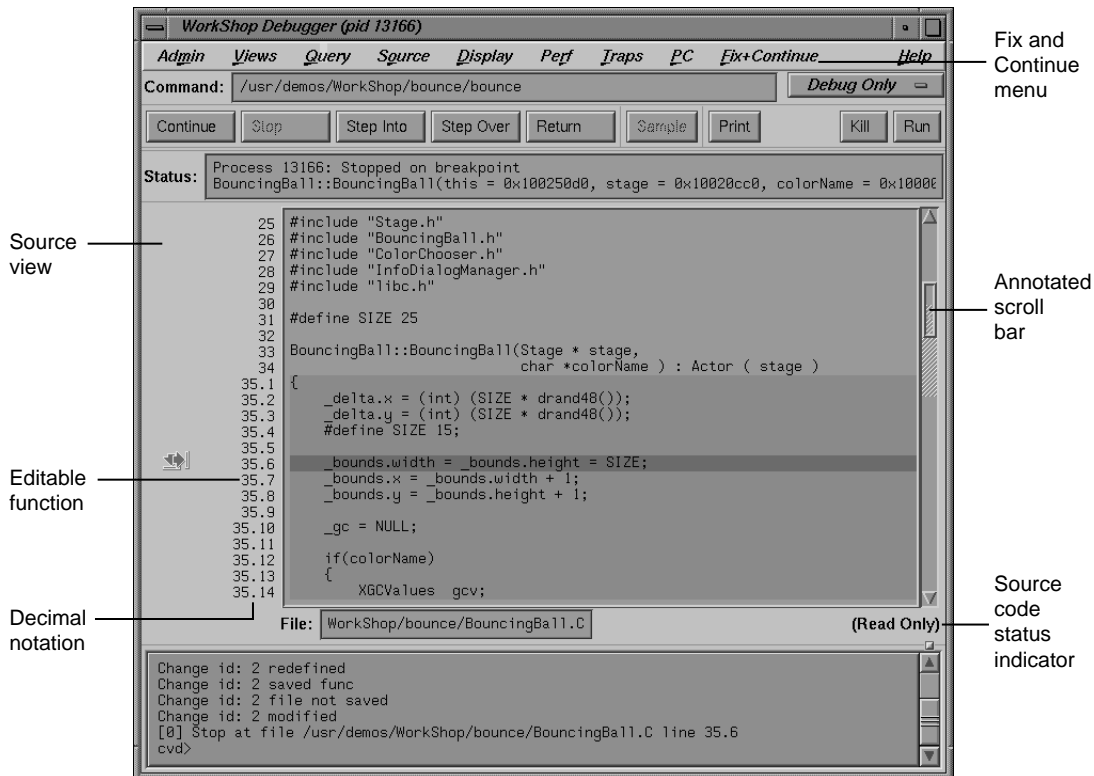
Figure 8-3 Debugger Main View Window

You can select 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+Continue commands and reports status involving redefined functions or files. Figure 8-4, page 107, shows a function

successfully redefined using the command line. Change id 1 was previously redefined and assigned the number 1.

Specify function with —— Change id 1

```
cvd> redefine 1
"/d2/people/cgeary/interp/fermat.c":18.1> {
"/d2/people/cgeary/interp/fermat.c":18.2> int i;
"/d2/people/cgeary/interp/fermat.c":18.3> int result = 1;
"/d2/people/cgeary/interp/fermat.c":18.4> for (i = 0; i < n; i++)
"/d2/people/cgeary/interp/fermat.c":18.5> result *= a;
"/d2/people/cgeary/interp/fermat.c":18.6> return result;
"/d2/people/cgeary/interp/fermat.c":18.7> }
"/d2/people/cgeary/interp/fermat.c":18.8> .
Change id: 1 redefined
         1        enabled /d2/people/cgeary/interp/fermat.c      power
Change id: 1 activated
cvd>
```
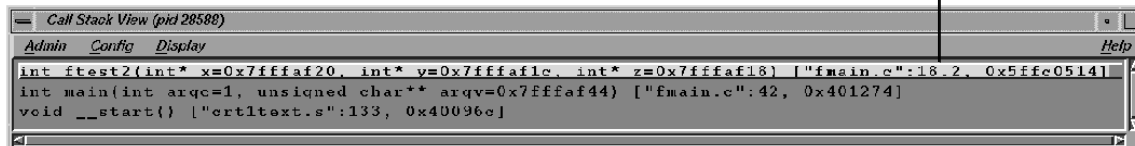
Figure 8-4 Command Line Interface with Redefined Function

Call Stack

The **Call Stack** recognizes redefined functions. It uses the decimal notation for line numbers, as shown in Figure 8-5.

Decimal notation for line number

```
Call Stack View (pid 28588)
Admin   Config   Display                                                Help
int ftest2(int* x=0x7fffaf20, int* y=0x7fffaf1c, int* z=0x7fffaf18) ["fmain.c":18.2, 0x5ffc0514]
int main(int argc=1, unsigned char** argv=0x7fffaf44) ["fmain.c":42, 0x401274]
void __start() ["crt1text.s":133, 0x40096c]
```

Figure 8-5 Call Stack

Trap Manager

The **Trap Manager** recognizes redefined functions. It uses the decimal notation for line numbers, as shown in Figure 8-6, page 108.
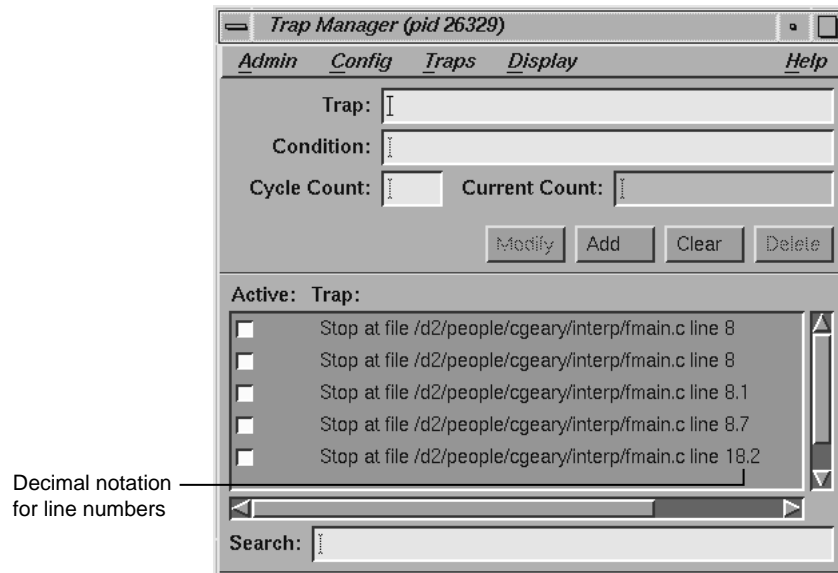
Figure 8-6 Trap Manager Window with Redefined Function

Debugger Command Line

You can debug programs by entering dbx-style commands at the cvd> prompt found at the bottom of the Main View window (see Figure 1-1, page 2). For more information, refer to the *dbx User's Guide*.

In the following text, **F+C** indicates that the command also applies to Fix+Continue code. In F+C code, line numbers are expressed in decimal notation (for example, 5.3 indicates that line 5 is the start of a function in the original source code and 3 is line 3 of that function in F+C). See Chapter 8, "Fix+Continue Windows", page 99 and information about Fix+Continue in the *ProDev WorkShop: Debugger User's Guide* for more details.

Note: Only applications compiled with −32 and −n32 options can use Fix+Continue features.

Syntax for Commands

The syntax for the debugging commands is as follows:

add_source [*filename*: [*line_number*] [,*line_number*]]

> **F+C**
>
> For C and C++ only, when compiled with −o32.
>
> 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 and replace_source.

alias [*shortform command*]

> Lists all aliases without arguments. With arguments, it assigns *command* to *shortform*.

assign *expression1=expression2*

> Assigns *expression2* to *expression1*.

attach *pid*

> Attaches to specified process ID (*pid*).

call function_name[*argument*, *argument*]

> Executes the specified function with any arguments supplied. The syntax of the call should be that of the source language of the application in use, such as C, C++, or Fortran.

catch [*signal_name* | all]

> With no arguments, lists signals to be breakpointed. If a signal is specified, it is added to the list. If all is specified, the debuggert traps all signals.

clearcalls

> Cancels interactive function calls that are currently active (that is, interactive function calls that are stopped on a breakpoint or watchpoint).

cont in *function_name*

> Continues execution from the current line to the entry to the specified function.

cont to *line_number*

> Continues execution from the current line until the specified line, if there is not an intervening breakpoint to stop execution.

continue [*all*]

> Continues executing a program, or all processes, after a breakpoint. You can use both c and cont as aliases for continue.

continue [*signal*]

> Sends specified signal and continues executing a program after a breakpoint.

corefile [*filename*]

> With no arguments, reports whether data referencing commands reference a core file. If so, displays the current core file. With *filename* provided, specifies core file to be debugged.

delete [all | [*bp_number*] ,[*bpp_number*], ...]

> Deletes all breakpoint(s). The all option deletes all breakpoints.

delete_changes [[*func_spec* | -all] | [-file *filename*]]

> **F+C**
>
> For C and C++ only when compiled with -o32.
>
> Deletes changes corresponding to the selected functions (for example, delete_changes getNumbers -file fmain.c). Once IDs are deleted, you cannot use the IDs again because the IDs associated with the selected functions are released. The default is -all. See also save_changes.

delete_source [filename: [*line_number*] [,*line_number*]]

> **F+C**
>
> For C and C++ only, when compiled with -o32.
>
> 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 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.

delete *bp_number* [,*bp_number*, ...]

> Deletes the specified breakpoint numbered *bp_number*, as obtained from the status command.

detach

> Detaches from the current process.

disable all

>   Deactivates all traps.

disable_changes [[*func_spec* | -all] | [-file *filename*]]

>   **F+C**
>
>   For C and C++ only, when compiled with -o32.
>
>   Disables specified changes for 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.

disable *bp_number* [,*bp_number*, ...]

>   Deactivates the specified breakpoint numbered *bp_number*, as
>   obtained from the status command.

display [*expression*, ...]

>   With *expression*, adds *expression* to the list of expressions displayed
>   whenever the process stops. With no arguments, lists all expressions.
>   See undisplay to delete an expression.

down [*number*]

>   Moves down the specified number of frames in the call stack. down
>   moves away from the direction of the caller toward the most deeply
>   nested function called..

dump

>   Prints local variable values in the visible scopes in the currently active
>   function..

enable all

>   Reactivates all inactive breakpoints.

enable_changes [*func_spec* | -all] | [-file *filename*]

>   **F+C**

For C and C++ only when compiled with `-o32`.

Enables specified changes for 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.

enable *bp_number* [[, *bp_number*, ...]]

Reactivates the specified breakpoint numbered *bp_number*, as shown by the `status` command.

expression/[*count*] [*format*] or *expression*, [*count*] / [*format*]

Prints the contents of the memory address specified by *expression*, according to the specified format. *count* represents the number of formatted items. The following format options are available:

d	Prints a 16-bit word in decimal.
D	Prints a 32-bit word in decimal.
o	Prints a 16-bit word in octal.
O	Prints a 32-bit word in octal.
x	Prints a 16-bit word in hexadecimal.
X	Prints a 32-bit word in hexadecimal.
b	Prints a byte in octal.
c	Prints a byte as a character.
s	Prints a string of characters that ends in a null byte.
f	Prints a single-precision real number (32-bit floating point).
g	Prints a double-precision real number (64-bit floating point).

file [*filename*]

Displays the name of the current or specified file (*filename*). If a file is specified, it becomes the current file.

func [*func_name*]

> Moves to the source code corresponding to the specified frame in the call stack or to the function in the executable if not on the stack.

givenfile [*filename*]

> With no arguments, displays name of current object file. With *filename*, specifies the executable to be debugged.

goto *linenumber*

> Skips over lines going directly to the specified line number in the current routine. Unlike dbx(1), cvd(1) does not begin execution at the specified line.

help

> Displays syntax information for the cvd command.

ignore[*signal_name* | all]

> With no arguments, lists those signals not to be breakpointed/trapped. If a signal is specified, this command removes it from the list of signals to be breakpointed/trapped. If all is specified, ignores all signals.

kill[*pid* | all]

> Kills the specified process currently controlled by the Debugger or kills all processes.

list [*from-line*[:*line_count*]][,*to_line*]|[*function_name*]

> Lists source lines beginning at line number *from-line*. If no additional argument is specified, the default for *line_count* is 10. If *line_count* is specified, a total of *line_count* lines are listed. If *function_name* is specified, the lines from the given function are listed.

list_changes [*func_spec* | -all] | [-file *filename*]

> **F+C**
>
> 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.

next [*int*]

> Steps over the specified number of source lines. This command does not step into procedures. The default is one line.

nexti [*int*]

> Steps over the specified number of machine instructions. This command does not step into procedures. The default is one instruction.

print *expression* [,*expression*, ...]

> Prints the value of the specified expression(s). If the expression is a character pointer or array, both the string and address print. Uuse p as an alias.

printd *expression* [,*expression*, ...]

> Prints the value of the specified expression(s) in decimal format. Use pd as an alias.

printf *string* [,*expression1* [, *expression2,*]...]

> Prints the value(s) of the specified expression(s) in the format specified by the *string* string. The printf command supports all formats of the IRIX printf command except %s. For a list of formats, see the printf(1) man page.

printo *expression* [,*expression*, ...]

> Prints the value of the specified expression(s) in octal format. You can use po as an alias.

printregs

> Prints the contents of all of the registers.

printx *expression* [,*expression*, ...]

> Prints the value of the specified expression(s) in hexadecimal format. Use px as an alias.

privateProject

> Used when running multiple cvd sessions simultaneously. This prevents one cvd session from being confused with anotoher cvd session.

pwd

> Displays the current directory.

quit

> Exits the debugging session.

redefine [*func_spec* [-edit | -read] *filename* [*line_number*,*line_number*]]

> **F+C**
>
> For C and C++ only.
>
> 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.
>
> - -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> .
```

replace_source [*filename*: [*line_number*] [,*line_number*]]

        F+C

        For C and C++ only when compiled with -o32.)

        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.

rerun

        Runs the program again using the same arguments.

return

        Continues executing the current procedure and returns to the next sequential line in the calling function.

run [[all] | *argument_list*]

        Runs the program(s). If an *argument_list* is specified, it is used as the arguments to be supplied to the program.

runtime_check *func_spec* [-options *key* [*key*,...]]

        F+C

        For C and C++ only when compiled with -o32.

Enables all run-time checking options by default. If `-options` is specified, then run-time checking is restricted to the *keys*. The result of the runtime checks are printed the next time the specified function (*func_spec*) is entered. You can invoke this command on a stopped or a running process.

save_changes {*func_spec*| {`-file` *filename*}} [-[w | a]] *filename_to_save*

F+C

For C and C++ only when compiled with `-o32`.

Saves (enabled or disabled) function redefinitions or an entire file to another file (*filename_to_save*). The following example shows how to save a function definition:

```
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
```

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

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

See also `delete_changes`.

setbuildenv [*filename*] compiler-flag-list

F+C

For C and C++ only, when compiled with `-o32`.

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`.

sh [*shell_command*]

Calls a shell if no arguments; otherwise, executes the specified shell command.

showbuildenv [*filename*]

F+C

For C and C++ only, when compiled with -o32.

Lists all the build environment flags set. `showbuildenv` with a *filename* lists any build environment specifications that have been set separately with `setbuildenv "`*filename*`"`.

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

F+C

For C and C++ only, when compiled with -o32.

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*]]

F+C

For C and C++ only, when compiled with -o32.

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).

showthread [full] [thread] [*number* | all]

> Shows brief status information about threads. If full is specified, prints full status information. You can request status information for a specific thread by number or you can request information for all threads. The thread qualifier does not affect command output. It is effectively a 'noise' word here, though allowed for consistency with other commands where the thread qualifier is needed to specify a particiular thread.

source *filename*

> Executes commands in the specified file as if those commands were typed on the command line..

status

> Displays a list of currently set breakpoints and traces.

step [*int*]

> Steps the specified number of source lines. This command steps into procedures. The default is one line.

stepi [*int*]

> Steps the specified number of machine instructions. This command steps into procedures. The default is one instruction.

stop [all | pgrp] in [*filename*]

> If set to all, stops all members of the specified process group whenever the trap is encountered. If set to pgrp, all members of the process group apply the trap.

stop at [*file*] [*filename*] [*line*] [*line_number*] [if *expression*]

> Sets a breakpoint at the specified line in the specified file. If the if option is used, the breakpoint fires only if *expression* is true. If specified, the file name must be enclosed in double-quotation marks. As an example, to specify a stop at line 5 in myfile.c, the syntax is:
>
> stop at "myfile.c":5

```
stop exception [all | item]}
```

Sets a breakpoint on all C++ exceptions or exceptions that throw the base type `item`. Do not include complex expressions using operators such as `*` and `&` in your type specification for an exception breakpoint.

```
stop exception [all | [item] [, item]]
```

Sets a breakpoint on all C++ exceptions or exceptions that either have no handler or are caught by an unexpected handler. If you specify `item`, stops on exceptions that throw the base type `item`. Do not include complex expressions using operators such as `*` and `&` in your type specification for an exception breakpoint.

`stop in` [*filename*`:`] *function_name* [`if` *expression*]

Sets a breakpoint at the entry to the specified function. If the `if` option is used, then the breakpoint fires only if *expression* is true. If the *filename* is given, the function is assumed to be in that file's scope. If specified, the *filename* must be enclosed in double-quotation marks. As an example, to specify a stop in function `func1` in `myfile.f` only if *n* is 20, the syntax is:

```
stop in "myfile.f":func1 if n .eq. 20
```

`syscall` [`catch` | `ignore`] [`call` | `return`] [*sys_call_name* | `all`]

The `catch` option adds a system call to the list of system calls to be breakpointed. The `ignore` option removes a system call from the system call breakpoint list. The `call` option specifies the entry to the system call and `return` signifies the return from the call.

`trace` [[[*variable*] `at` *filename*] | *line_number* | *function_name*] [`if` *expression*]

Traces the specified variable. You can specify a file and/or test condition. You can also specify a line number or a function where the trace is to take place. If *expression* is allowed, only a variable, line number, or function name is typed (that is, `trace if ==3` is not allowed).

`unalias` *aliasname*

Cancels the alias specified as *aliasname*.

undisplay [[*displaynumber*, ...]

> Stops display of expression with specified *displaynumber* when the process stops. Removes the expression from the display list.

unsetbuildenv[*filename*]

> **F+C**
>
> For C and C++ only, when compiled with -o32.
>
> 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.

up [*number*]

> Moves up the specified number of frames in the call stack. up moves in the direction of the caller.

use [*path*]

> Uses the specified path to search for source files.

watch *identifier* [write | read]

> Causes program to stop when the *identifier* is written or read, depending on whether write or read is specified. If neither is specified, the default is write.

whatis *identifier*

> Displays the type information of the specified variable or function.

when at [*filename*] [*line_number*] [*command*] [*command*] ...

> When your program reaches the specified *line_number*, the commands specified are executed (the program resumes execution automatically).

when in [*filename*] *function_name* [*command*] [*command*] ...

> When your program enters the specified function, the commands specified are executed before the program resumes execution.

which [*identifier*]

> Displays the qualification of the specified variable.

where [thread | *thread-id*] [*n*]

> Performs a stack trace showing the activation levels of a program or, optionally, of the specified thread. You can obtain *thread-ids* from the first column of output of the showthread command. *n* is the number of levels for the output.

Blocking Kernel System Calls

The following are the kernel system calls (syscalls) that actually block continued pthreads. There are numerous library routines, such as printf, that can use one of these blocking system calls.

It would be impractical here to list all library routines which utilize a blocking syscall. Nevertheless, as a user you should know, for example, that if you call the printf library routine it eventually calls writev(), a blocking system call, and thus may block continued pthreads.

accept

> accept a connection on a socket

close

> close a file descriptor

creat

> create a new file or rewrite an existing one

dmi

> SGI specific. Used to implement the interface defined in X/Open document *Systems Management: Data Storage Managment (XDSM) API*.

fcntl

> File and descriptor control. Provides for control over open descriptors.

fsync

> synchronize a file's in-memory state with that on the physical medium

getmsg / getpmsg

> get next message off a stream

ioctl

> Control device. Performs a variety of control functions on devices and streams.

lockf

> Record locking on files. Allows sections of a file to be locked.

mq_open

> open/create a message queue

msgsnd / msgrcv

> message send and message receive

msync

> synchronize memory with physical storage

nanosleep

> high resolution sleep

open

> open for reading and writing

pause

> suspend process until signal is received

poll

> input/output multiplexing

putmsg / putpmsg

> send a message on a stream

read / readv / pread

> read from a file

recv / recvfrom / recvmsg

> receive a message from a socket

select

> synchronous I/O multiplexing

```
semget / semctl / semop
```

        semaphore handling

```
send / sendto / sendmsg
```

        send a message from a socket

```
sginap
```

        times sleep and processor yield function

```
write / writev / pwrite
```

        write on a file

Index

Tree, selection in Data Explorer display
    submenu, 82
Type color, field in Data Explorer tydpe
    formatting box, 85
Type name, field in Data Explorer type
    formatting box, 84

U

unalias dbx-style command, 122
undisplay dbx-style command, 122
unsetbuildenv, dbx-style command, 122
up, dbx-style command, 122
Update, selection in Data Explorer display
    menu, 82
use, dbx-style command, 122

V

Variable browser
    general description, 86
Variable Browser, selection in views menu, 9
variables
    compiled with -n32 -g, 86
    unused, 86
Versioning, selection in Source menu, 11
view changes in debugger, 105
View menu, 100
View submenu, 15
view, call stack, 107
Views menu in the Main View window, 8

W

Warn Unfinished Edits Before Continue, toggle in
    Fix+Continue Preferences dialog, 16
Warn Unfinished Edits Before Run, toggle in
    Fix+Continue Preferences dialog, 16
watch, dbx-style command, 122

whatis dbx-style command, 122
when at, dbx-style command, 123
when in, dbx-style command, 123
where, dbx-style command, 125
which, dbx-style command, 123
widget class examiner, 53
widget class menu item, 36
widget examiner, 49
widget item, 36
widget tree menu item, 36
window examiner, 52
Wrapped display, selection in array browser
    display menu, 63
write syscall that blocks continued pthreads, 127
writev syscall that blocks continued pthreads, 127

X

X event menu item, 36
X graphics context menu item, 36
X pixmap menu item, 36
X-event breakpoints examiner, 45
X/Motif Analyzer
    global objects description, 35
X/Motif Analyzer admin menu, 35
    close, 36
    save as text, 36
X/Motif Analyzer breakpoint type option
    button, 39
X/Motif Analyzer breakpoints examiner, 38
    callback, 40
    event-handler, 41
    input-handler, 43
    resource-change, 42
    state-change, 44
    timeout-procedure, 42
    X-event, 45
    X-Request Breakpoints Examiner, 46
X/Motif Analyzer callback examiner, 52
X/Motif Analyzer event examiner, 52