

# **HP WDB-GUI Reference Manual**

**HP-UX 11i v2**

**Edition 3**



**i n v e n t**

**Manufacturing Part Number: 5991-1265**

**September 2005**

Printed in the United States

© Copyright 2004-2005 Hewlett-Packard Development Company, L.P.

---

## Legal Notices

The information in this document is subject to change without notice.

*Hewlett-Packard makes no warranty of any kind with regard to this manual, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose.* Hewlett-Packard shall not be held liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

### Warranty

A copy of the specific warranty terms applicable to your Hewlett-Packard product and replacement parts can be obtained from your local Sales and Service Office.

### U.S. Government License

Proprietary computer software. Valid license from HP required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

### Copyright Notice

© Copyright 2004-2005 Hewlett-Packard Development Company, L.P. Reproduction, adaptation, or translation of this document without prior written permission is prohibited, except as allowed under the copyright laws.

### Trademark Notices

UNIX® is a registered trademark in the United States and other countries, licensed exclusively through The Open Group.

<b>1. WDB GUI Overview</b>	
Summary of Features . . . . .	9
<b>2. Installing WDB GUI</b>	
Installation Overview . . . . .	16
Supported Configurations . . . . .	16
Disk Space Requirements . . . . .	16
Installation Instructions . . . . .	16
Filesets . . . . .	16
<b>3. Debugging with WDB GUI</b>	
Loading a Program and Changing Program Settings . . . . .	20
Setting Run Options . . . . .	21
Setting Environment Variables . . . . .	22
Setting Program Input/Output Redirection . . . . .	24
Starting and Stopping the Debug Process . . . . .	27
Starting the debug process (Go) . . . . .	27
Restarting the debug process (Restart) . . . . .	27
Stopping a debug session . . . . .	28
Pausing program execution . . . . .	28
Using Breakpoints . . . . .	29
Tip . . . . .	30
Inserting breakpoints . . . . .	31
Removing breakpoints . . . . .	31
Viewing and modifying breakpoints . . . . .	32
Working with deferred breakpoints . . . . .	33
Advancing Through Your Program . . . . .	35
Stepping into functions . . . . .	35
Stepping over functions . . . . .	35
Stepping out of functions . . . . .	36
Stepping last functions . . . . .	36
Running to the cursor . . . . .	37
Setting the next statement to execute . . . . .	37
Showing the next statement to execute . . . . .	38
Setting Signal Handling . . . . .	39
Viewing signal handling . . . . .	40

Changing signal handling . . . . .	40
Browsing Functions . . . . .	42
Viewing source for a function . . . . .	42
Specifying a function filter . . . . .	42
Listing and viewing member functions . . . . .	43
Using the Program Console . . . . .	44
Viewing program output . . . . .	44
Entering program input . . . . .	44
Changing the Working Directory . . . . .	46
To set the current working directory . . . . .	46
Setting source paths . . . . .	47
Adding and deleting source paths . . . . .	47
Changing the order of source paths . . . . .	48
Setting Object Paths . . . . .	50
Adding and deleting object paths . . . . .	50
Changing the order of object paths . . . . .	51
Setting Pathmap . . . . .	53
Adding and Deleting Pathmap . . . . .	54
Changing the Order of Pathmap . . . . .	55
Opening Source Files . . . . .	56
To open a source file . . . . .	56
To reopen a source file that was recently open . . . . .	56
To save source files . . . . .	56
To close a source file . . . . .	57
Viewing Code . . . . .	58
Viewing source code . . . . .	58
Viewing assembly code . . . . .	60
Viewing variables using Dwell . . . . .	61
Fixing the Code from within the Debugger . . . . .	63
Fixing code and continuing your program . . . . .	64
Explicitly requesting the debugger for monitoring or rebuilding files . . . . .	65
Examining build errors and warnings . . . . .	65
Restrictions on fixing code from within the debugger . . . . .	65
Finding Specific Text in Your Code . . . . .	67
To find text or a regular expression . . . . .	67
To find the next match . . . . .	68

Saving and Restoring Debug Sessions . . . . .	69
To save a debug session . . . . .	71
Restoring a debug session . . . . .	71

#### **4. Troubleshooting WDB GUI**

Error Messages. . . . .	74
Troubleshooting Tips . . . . .	77
Debugging executables over NFS . . . . .	77
Attaching to a process with shared libraries. . . . .	77
Double-Clicking with ReflectionX . . . . .	77
Setting breakpoints that never appear in the breakpoints list . . . . .	77
Problems with keyboard shortcuts or mnemonics . . . . .	78
Using Online Documentation . . . . .	79
Setting up a Web browser to view Help. . . . .	79
Locating information . . . . .	79
Understanding the navigation icons . . . . .	80
Printing Topics . . . . .	80
Getting Support . . . . .	81



---

# 1 WDB GUI Overview

The HP WDB GUI is a Graphical User Interface (GUI) designed by Hewlett-Packard for the WDB debugger. WDB stands for Wildebeest Debugger, which is HP's implementation of the GDB debugger. WDB GUI supports debugging of 32-bit programs on HP-UX 10.20 and both 32-bit and 64-bit programs on HP-UX 11.x on PA systems, and also runs on HP-UX 11.22

Itanium systems. WDB GUI debugs native-compiled HP C, HP aC++, and Fortran programs.

Using WDB GUI, you can do the following:

- Debugging an executable file or a core file produced by an aborted program
- Attaching to a process that is already running
- View your source code and the corresponding assembly code
- Edit breakpoints
- Watch local and global variables
- Traverse the call stack
- Modify signal handling
- View registers
- Switch between threads in your program
- View Heap usage and Memory leaks report
- Edit sources and have the debugger compile and patch in the changes without restarting the debugged program.

You can directly execute the WDB commands in the command view. WDB also allows you to save your debug session and restore it later.

WDB GUI is similar to the PC-based debuggers in its look, feel, and navigation methods. In a multi-platform environment, WDB GUI provides seamless transition between your PC development tools and the HP WDB debugger.



## Summary of Features

WDB GUI includes the following features:

- Support for the `-mapshared` option: WDB GUI 5.2 supports the `-mapshared` functionality. This option suppresses the mapping of all shared libraries in a process private. The `-mapshared` option can be given as a command-line option while invoking WDB GUI as:

```
$ wdb -mapshared
```

or in the `gdb` command prompt as:

```
(gdb) set mapshared on
```

To allow shared libraries to be loaded after the current point to be mapped private:

```
(gdb) set mapshared off
```

- Enhanced Threadview : The threadview support has been enhanced to mark the last-event thread. This feature does the bookkeeping for the thread that was running before a `ttrace` event, which stopped the process being debugged. In cases when both the current thread and last-event thread are the same, only the current thread is flagged.
- Save To File option:  
The Save to File option is used to save the contents of debugger stateview to the file specified. Right click on the debugger stateview, to save the view contents, to a file.
- Enhanced Registers View : WDB GUI register view has been enhanced to show double precision floating-point register values. For example :  
Single precision registers - `fr7L = 26.1749401 fr7R = 0`  
Double precision register - `fr7 = 1.5000000000000016`
- Enhanced support for debugging terminal user interface based applications : WDB GUI supports a new command-line option `-tui` to support debugging terminal user interface based applications (for example `vim`).
- Customized display of Program Console: WDB-GUI supports a command line option `-d` to customize WDB GUI Program Console display.
- Enhanced Dwell feature to show large data: The Dwell feature has been enhanced to display large data.
- Enhanced Source View: WDB GUI will be consistent in displaying source files in the Source View for FORTRAN applications.

- No dwelling on Code comments: WDB GUI is now able to distinguish Code comments and will not show the value when dwelled on it.
- Support for WDB's Pathmap functionality: WDB GUI supports HP WDB debugger's Pathmap functionality. The Pathmap 'From' and 'To' dialog box lets you define a list of substitution rules to be applied to path names to identify object files and the corresponding source files. In this dialog box, you can add, delete, and change the priority order of pathmap. You can access the Pathmap dialog box from the Edit menu by clicking Pathmap. Alternately, you may also set the pathmap from WDB GUI's command view.
- Setting Object File Paths: This feature allows the user to set object file paths so that WDB GUI may locate object files for the current program. The user may enter object file paths either via WDB GUI's command view or via the Object File Paths Dialog box. The Object File Paths dialog box may be accessed on the Edit menu by clicking Object File Paths.
- Support for Alternate Root Functionality: WDB GUI supports HP WDB debugger's Alternate root path functionality. The Alternate Root Path functionality enables the user to install HP WDB debugger and its components on an alternate root rather than the system default root. HP WDB provides an environment variable, GDB\_ROOT for this purpose. WDB GUI facilitates the setting of this environment variable. If GDB\_ROOT is not set and the environment variable WDB\_ROOT is set, WDB GUI would set GDB\_ROOT to the WDB\_ROOT value. If GDB\_ROOT is set, the value is left as such.
- Support for Debugging PA-RISC Applications on Itanium-based Systems: This feature allows the user to transparently debug PA-RISC applications and core files on Itanium-based systems in compatibility mode under Aries. When an executable is loaded and if the debug target is a PA-RISC binary then WDB GUI automatically changes the debugger flavor to PA-RISC version of HP WDB.
- Enhanced Array Browsing: Allows you to view the entire contents of an expandable expression with a single click on the expression's name, in the Name field of the Local Variables /Watch views. The expandable expression can be an array, structure, union or a class. To view the contents of the expandable data structure, click on the plus sign (+) available on the left side of the data structure's name, or by double clicking the Name field on the line. To collapse the whole expansion, click on the minus sign (-) or double click on the line again. If the Enhanced Array Browsing option is set, the Quick Watch view will show the entire contents of the expandable data structure by default. To set this option, check the Enhanced Array Browsing check box in the Edit > Preferences dialog box. If this option is not set, you can click an expression's name to expand that expression to the next level.

---

**NOTE**                    The expression is of pointer data type, it is expanded only to the next level.

---

- **Memory Log:** The Memory Log feature enables you to save the reports that the Memory Usage view creates about the data that the debugger collects on your program's memory leaks and heap use.
- **Enable/Disable Thread:** While debugging a multithreaded application, if you suspect that a specific thread is causing a problem, suspend other threads in the debugger and debug the thread that causes the problem. The Enable/Disable Thread feature in WDB GUI enables you to disable and enable specific threads.
- **Steplast support for C and C++:** This feature, `steplast`, will step into a function, but not into the calls for evaluating arguments. In general, if a function call has arguments, which makes further function calls, the step command will step into argument evaluation call.

---

**NOTE**                    The `Steplast` feature is supported in WDB GUI with HP WDB version 3.1.5 or higher.

---

- WDB GUI allows you to configure the number of string characters or array elements to be displayed in the LocalVariables/Watch/QuickWatch views. Lower values can mean less memory is consumed while displaying large data structures. The default value is 200. You can set the value to a number between 0 and 1700 in the "Maximum Array Elements to be printed" text box using `Edit > Preferences` menu. However, to optimize memory consumption of WDB GUI, it is recommended that you keep this number as small as needed for your debugging session.
- **Shell commands:** WDB GUI provides a separate popup terminal window for the user to execute shell commands. Type `exit` in the popup window to close the popup window and return the prompt to WDB GUI. If no `TERM` value is specified, `xterm` is taken for popup by default.
- **Hexadecimal display in Dwell:** This feature allows you to view the hexadecimal values of the variables. When you dwell on a variable, the value of the variable is displayed in decimal and hexadecimal formats as a tooltip in the Source view of the WDB GUI. To view only decimal values, disable the hexadecimal display by setting the preference in "Display in decimal & hexadecimal format with dwell feature" checkbox in the Debugger Preferences dialog box.
- WDB GUI loads large data structures faster than before in the Local Variables/Watch/Quick Watch views. The load time of large size arrays, structures, and classes is much shorter than before.

---

**NOTE** WDB supports WDB GUI performance improvement. It is recommended that you use the tuned WDB GUI 3.1.1 with WDB version 3.1.1. Earlier versions of WDB might not show good improvements with WDB GUI 3.1.1 and might show unexpected behaviour with respect to the Local Window/ Watch Window/Quick Watch Window.

---

- The `Print` debugger command: This command has been added to the source popup menu. You can mask any variable and right click (click M2) on this source popup menu item to print the value of the highlighted variable in the Command view.
- Saving Commandline History: Allows you to save commandline history across WDB GUI sessions. You can also save and restore the sequence of commands in the GUI commandline, along with the other session contents, using `Save/Restore Session`.
- Creating Buttons Dynamically: Allows you to create buttons dynamically from within the WDB GUI. You can associate all operations that are valid at the debugger prompt (`gdb`) with a button. You can save and restore the list of dynamically created buttons across GUI sessions.
- Automatic update of source search path when breakpoint is set: WDB GUI automatically updates the source search path when a breakpoint is set in a file that is not in the search path. You can set this by checking the "Automatically Update Search Path when breakpoints set" check box in the Debugger Preferences dialog box.
- Fix and continue: Allows you to edit sources and make the debugger compile and patch in the changes without restarting the debugged program.
- Memory checking: Allows you to configure the debugger to check and report heap use, memory leaks, and memory use errors.
- Function browsing: Allows you to visit the source for a function by clicking on the function name. You can choose from a list of functions that match a given regular expression.
- Point and click breakpoints and breakpoint editing. Allows you to set or delete a breakpoint by clicking M1 on the left of the Source view or Disassembly view. Click M2 on a breakpoint symbol to bring up a dialog that lets you modify that breakpoint's attributes.
- Program consoles: `xterm`, `dtterm`, `hpterm` are supported as program consoles
- Debugging an executable file or a core file produced by an aborted program
- Attaching to a process that is already running
- Debugging 32-bit programs
- Debugging 64-bit programs (HP-UX 11.x only)

- Viewing a source file and the corresponding assembly code
- Searching for specific text and regular expressions in several views
- Setting breakpoints and stepping through your program
- Viewing and modifying the values of local and global variables
- Viewing hardware registers
- Traversing the call stack
- Viewing and controlling signal handling
- Traversing threads
- Sending commands directly to WDB using a command line
- Accessing a history of commands you have executed during the current debug session
- Customizing the appearance and behavior of the debugger GUI
- Saving and restoring debugger sessions



---

## **2 Installing WDB GUI**

## Installation Overview

The following sections describe the requirement for installing WDB GUI:

### Supported Configurations

WDB GUI 5.2 is supported on HP-UX 11.x systems.

### Disk Space Requirements

**Table 2-1** Disk Space Requirements for Installing WDB GUI

Configuration	On 11.0 (PA)	On 11.22 (IA)
WDB-GUI only (WDB already installed)	8 MB	8 MB
WDB and WDB-GUI	16 MB	16 MB

### Installation Instructions

WDB GUI requires HP WDB. If you do not have HP WDB 5.2 installed on your system, it will be loaded automatically when you install WDB GUI.

Use the `SD-UX swinstall` command to install your software. It invokes a user interface that leads you through the installation. It also gives you information about disk space requirements, version numbers, product descriptions, and dependencies.

---

**NOTE** When you install the WDB GUI with `swinstall`, select both the WDB GUI and WDB products. If you select only the WDB GUI product, the WDB documentation does not load automatically.

---

For more information about `swinstall`, refer to the `sd(5)` man page, the `swinstall(1m)` man page, or the online help in `swinstall`.

### Filesets

The WDB-GUI product contains the following subproducts:



- Runtime contains one fileset: WDB-GUI-RUN
- Help contains one fileset: WDB-GUI-HELP
- Manuals contain one fileset: WDB-GUI-MAN

---

**NOTE**

Selecting the WDB GUI product will automatically select only the Runtime subproduct of the WDB product. When you install the WDB GUI with `swinstall`, select both the WDB GUI and WDB products. If you select only the WDB GUI product, the WDB documentation does not load automatically.

---



---

## **3 Debugging with WDB GUI**

---

## Loading a Program and Changing Program Settings

You can set the following options for your debug sessions:

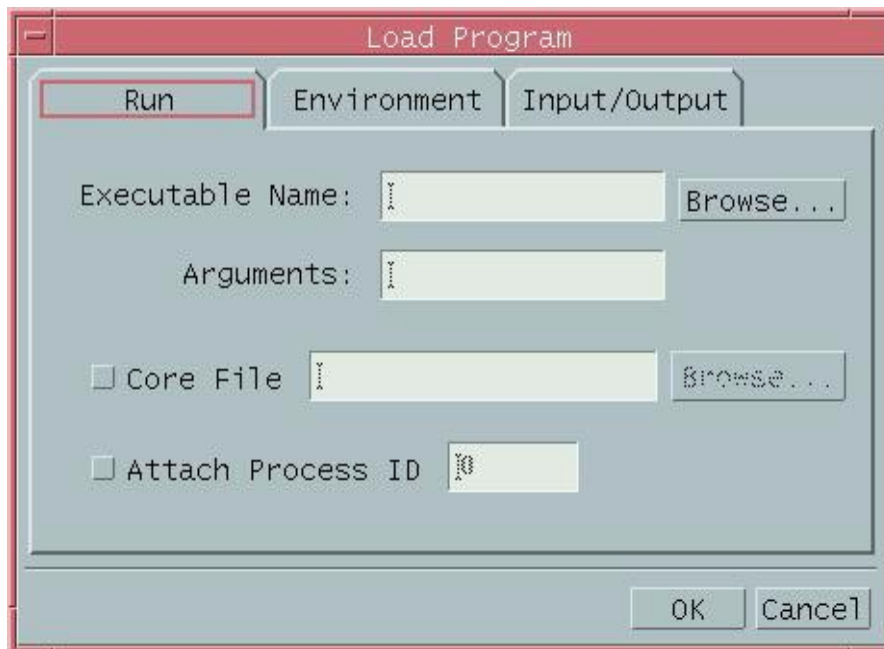
- Run options
- Environment variables
- Standard input/output redirection

---

**NOTE** When you click OK in the Load Program dialog box, any settings that you have specified in the Command view will be overridden.

If you change any debug settings or click OK in the Load Program dialog box, WDB reloads your executable and unloads the running executable.

---



The Load Program dialog box with the Run tab selected

## Setting Run Options

In the Load Program dialog box, use the Run tab to set options that specify the executable name, program arguments, core file name, and existing process to be used for the current debug session.

### To specify an executable:

1. On the File menu, click Load Program, then click the Run tab.
2. In the Executable Name box, type the path and name of the executable. To browse for the file, click the Browse button.
  - a. If you browse for the file, the Executable Name dialog box displays
  - b. In the Executable Name dialog box, double-click to select the appropriate directory from the Directories list and select the appropriate file from the Files list.
  - c. When the proper path and file appear in the Selection box, click OK.
3. In the Load Program dialog box, verify that the proper path and file appear in the Executable Name box.
4. In the Arguments box, type the command line arguments that you want to pass to the executable when it runs.
5. Click OK.

### To load a core file for debugging

1. On the File menu, click Load Program, then click the Run tab.
2. Select the Core File option, then type the path and name of the core file. To browse for the file, click the Browse button.
3. Click OK.

### To attach to an existing process for debugging

1. On the File menu, click Load Program, then click the Run tab.
2. Select the Attach Process ID option, then type the process ID number (PID).
3. Click OK.

The process execution pauses and the program loads into the WDB GUI. You can now set breakpoints, view variables, and perform other debugging tasks.

Set desired breakpoints at this stage because once you continue executing the process, you will not have control of execution until a breakpoint has been reached.

---

**NOTE**

- You can debug executables that are stored on NFS-mounted file systems, but you cannot attach to an existing process if the executable that you run resides on an NFS-mounted file system rather than on your local machine's file system. To fix this problem, copy the executable onto your local machine's file system, restart the process on your local machine, run the debugger, and attach to the process.
- If you want to attach to a running process that contains shared libraries, before you run your program, run the command `/usr/bin/pxdb -s enable executable_file` where `executable_file` is the name of your program executable.

---

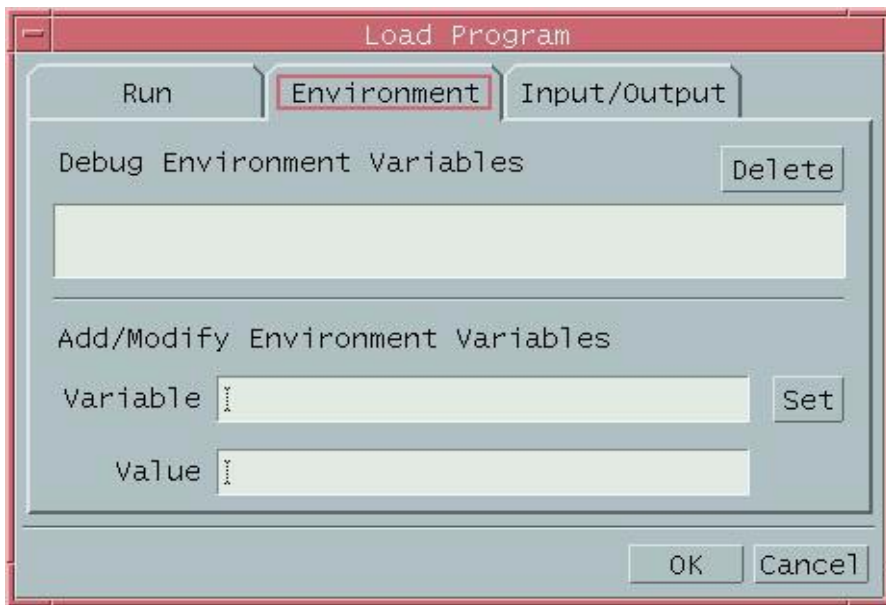
**To detach process**

Click Stop Debugging on the Debug menu to detach a process.

After the process is detached, the process continues running independently of the WDB GUI, and the WDB GUI no longer has any control of the process.

**Setting Environment Variables**

You can set environment variables that will be in effect for the current debug session.



### To view current environment variables

1. On the File menu, click Load Program, then click the Environment tab.
2. Check the Debug Environment Variables list box for a list of current environment variables.
3. Click OK.

---

### NOTE

- The Load Program dialog box does not display the environment variables that you specify using the Command view.
  - The Load Program dialog box does not display the environment variables that were inherited from the shell from which the WDB GUI was launched.
  - To see a list of all the environment variables that are currently set, use the `show env` command in the Command view.
-

### To set new environment variables

1. On the File menu, click Load Program, then click the Environment tab.
2. In the Variable box on the Add/Modify Environment Variable section, type the name of the variable you want to set.
3. In the Value box, type the value that you want to assign to the specified variable. For example, to assign the variable, “DISPLAY,” a value of “test:0,” type DISPLAY in the Variable box and type test:0 in the Value box.
4. Click Set.
5. Click OK.

### To delete environment variables

1. On the File menu, click Load Program.
2. Click the Environment tab.
3. In the Debug Environment Variables list box, select the variable that you want to delete.
4. Click Delete and click OK.
5. In the Command view, enter the command, `unset env Varname` where VarName is the name of the environment variable you want to remove.
6. Use the `show env` command in the Command view to verify that the environment variable has been removed.

## Setting Program Input/Output Redirection

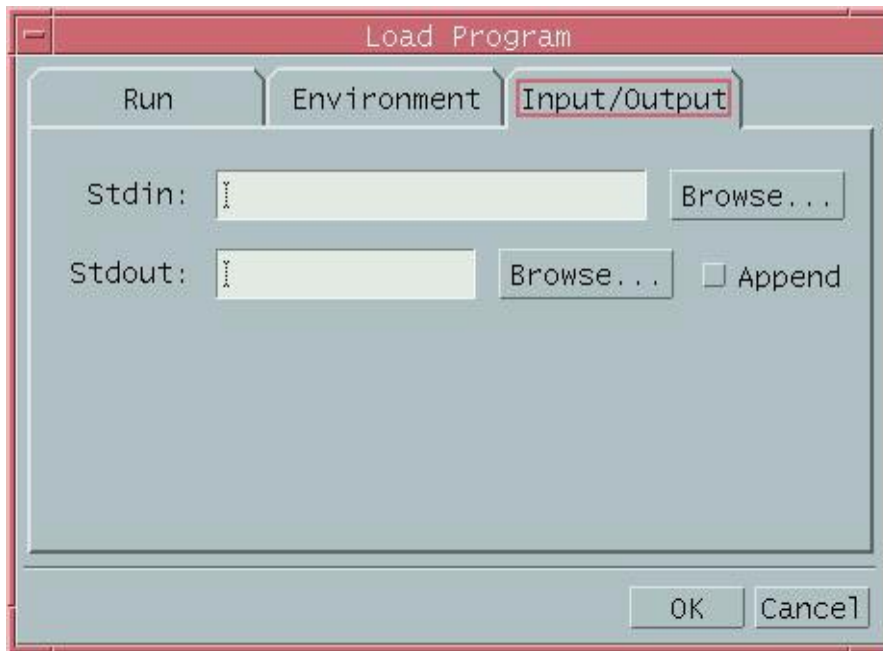
You can specify where you want to accept standard input and send output information for the current debug session. When these fields are blank, the Program Console is used for standard input and output.

---

**NOTE** Files you redirect using the Command view do not appear in the Load Program dialog box.

---





### To redirect standard input and output information

1. On the File menu, click Load Program.
2. Click the Input/Output tab.
3. In the Stdin box, type the name of the file or browse to the file that you want to specify for standard input.
4. In the Stdout box, type or browse to the file that you want to specify for standard output. Click the Append button if you want to append information to the standard output file rather than overwrite existing information in the file.
5. When you have the correct file name in each box, click OK.

---

**NOTE** You cannot redirect standard error from the WDB GUI. It will display in the Program Console. If you want to redirect standard error output, start your program in a terminal window outside the WDB GUI and then attach to the process. This allows you to control standard error output.

---

## Loading a Program and Changing Program Settings

**Tips:** To save the program settings that you set in the Load Program dialog box, click Save Session on the File menu and make sure that the Program Settings (Load Program) check box is checked.

## Starting and Stopping the Debug Process

You can use the commands in this section to control the execution of your program in the debugger. The Go and Restart options provide two ways to start the debugging process. For example, you may need to restart the debugging process as you use breakpoints to isolate problems in your program.

Stop Debugging offers a way to stop execution and enable loading of a new executable. If the debugger is attached to an existing process, Stop Debugging detaches the process. The Break Execution option allows you to interrupt the program while it is executing, which is helpful when you have not set breakpoints or cannot run to the cursor interactively. For example, you can use Break Execution to stop execution in an infinite loop.

### Starting the debug process (Go)

The Go option starts program execution or continues execution from the location of the program counter.

#### To start debugging

1. On the Debug menu, click Go.
2. Or, click the Go button in the toolbar:



### Restarting the debug process (Restart)

The Restart command resets execution to the beginning of the program being debugged. It uses the current program arguments and environment variables settings. The debugger pauses at the program's entry point, and clears and initializes the Watch, Local Variables, and Call Stack views.

If the debugger is attached to an existing process, Restart detaches the process and starts a new process.

#### To restart the debug process

1. On the Debug menu, click Restart.

2. Or, click the Restart button in the toolbar:



## Stopping a debug session

The Stop Debugging command stops executing the current program and enables loading of a new executable. The program is still loaded in the debugger, but no process is running. All views except the Source view are cleared and initialized.

If the debugger is attached to an existing process, Stop Debugging detaches the process. After it is detached, the process continues running independently and WDB no longer has any control of the process.

### To stop a debug session

1. On the Debug menu, click Stop Debugging.
2. Or, click the Stop Debugging button in the toolbar:



## Pausing program execution

The Break Execution command pauses execution of the program running in the debugger. Once the program is paused, you can continue debugging as if you had encountered a breakpoint at that location.

### To pause program execution

1. On the Debug menu, click Break Execution.
2. Or, click the Break Execution button in the toolbar:



**Tips:** Changing any debug settings will cause your executable to restart.

## Using Breakpoints

You can set breakpoints that pause program execution prior to the breakpoint location. In this way, you can evaluate variables, set breakpoints, view the call stack, or perform other debug actions at specific locations during the debug process.

In addition, you can specify the following:

- The commands that you want to execute when a breakpoint is reached
- The conditions when the breakpoint is active
- The number of times a breakpoint to be ignored before execution is paused

**Table 3-1 Breakpoint dialog box description**

Field/Button	Action
Break at text box	Type the location of the breakpoint you want to set. You can enter a line number for the file displayed in the Source view, file and line number in the form file:line, a function name, or an address in the form *address.
Stop at breakpoint only if the following expression is true text box	Type the conditions under which you want the debugger to break at the location in the Break at text box.
Enabled check box	Toggle to enable or disable a selected breakpoint.
Temporary check box	Check this box if you want the breakpoint to be removed when it is reached.
Enter WDB commands to invoke at breakpoint text box	Type a WDB command into the text box. This command will be executed each time the selected breakpoint is reached. Type one command per line.

**Table 3-1 Breakpoint dialog box description (Continued)**

<b>Field/Button</b>	<b>Action</b>
Number of breakpoint hits to ignore before stopping text box	Type the number of times you want to skip the selected breakpoint. This means that during execution, the WDB GUI will ignore the breakpoint “n” times and then pause at the breakpoint before executing “n+1.”
Breakpoints list	Displays all breakpoints that you have set. You can select a breakpoint that you want to perform an action upon.
OK button	Click to submit the information in this dialog box.
Cancel button	Click to exit this dialog box without submitting any of the information that is entered.
Goto Code button	Click to go to the location in the code where the selected breakpoint is set.
Remove All button	Click to remove all breakpoints.
Remove button	Click to remove the selected breakpoint.

---

**NOTE** The breakpoints you add, delete, or modify in this dialog box are not actually changed until you click OK.

---

### **Tip**

You can insert a breakpoint using the Breakpoints dialog box. You can also insert a breakpoint at the current cursor location in the Source or Disassembly view by doing one of the following:

- Right-click in either the Source or Disassembly view, and on the pop-up menu, click Insert Breakpoint.
- Or, click the toolbar button.

## Inserting breakpoints

### To insert a breakpoint

1. On the left side of the text in the Source or Disassembly view, place your cursor at the line where you want to insert a breakpoint.
2. Click M1. A breakpoint marker (a solid red octagon) appears in the left margin of the view, indicating that the breakpoint is set.

Or

1. In the text column of the Source or Disassembly view, place your cursor where you want to insert a breakpoint.
2. Right-click and click Insert Breakpoint. A breakpoint marker (a solid red octagon) appears in the left margin of the view, indicating that the breakpoint is set.

---

**NOTE** You may set a breakpoint in a file that is part of your executable but is not located in the source paths that you have specified. Add the appropriate source path in the Source File Paths dialog box.

---

## Removing breakpoints

1. In the Source or Disassembly view, place your cursor on the breakpoint symbol for the breakpoint you want to remove.
2. Click M1 to remove the breakpoint.

Or

1. On the Edit menu, click Breakpoints.
2. From the Breakpoints list in the Breakpoints dialog box, select the breakpoint you want to remove.
3. Click Remove. The selected breakpoint disappears from the list. To remove all the breakpoints that you have set, click Remove All.

4. Click OK.

## Viewing and modifying breakpoints

The Breakpoints dialog box allows you to view and modify breakpoint attributes. You can also use this dialog box to add and remove breakpoints.

### To bring up the Breakpoints dialog box

1. Place the cursor on a breakpoint symbol.
2. Click M2. The Breakpoints dialog appears with the breakpoint you clicked on selected.

Or

1. On the Edit menu, click Breakpoints.
2. You can use the dialog box to view, modify, add, and remove breakpoints.
3. Click OK to commit your changes and exit the dialog box.

### To view attributes of a breakpoint

Select a breakpoint. The Breakpoints list in the Breakpoints dialog box lists all the breakpoints that are set. Click on a specific breakpoint in the list to view its attributes

### To modify attributes of a breakpoint

1. Select a breakpoint. The Breakpoints list in the Breakpoints dialog box lists all the breakpoints that are set. Click on a specific breakpoint in the list to view its attributes.
2. You can modify any of the following attributes:
  - To change the breakpoint location, edit the text in the Break at area.

---

**WARNING**

**If you erase the text in the Break at text area, the breakpoint will be removed.**

---

- If you want to break only under a certain condition, enter that condition in the Stop at breakpoint only if the following condition is true text area.
- To enable or disable a breakpoint, toggle the Enabled check box to set the breakpoint as enabled or disabled. See Tips for shortcuts for enabling and disabling breakpoints.

---

**NOTE**





Indicates an enabled breakpoint.



Indicates a disabled breakpoint.

- 
- To break at a location only once, check the Temporary check box. Once a temporary breakpoint has been reached, it is removed automatically from the list.
  - To execute WDB commands when a breakpoint is reached, enter the commands in the Enter WDB commands to invoke at breakpoint text area. Type each command on a separate line.
  - To skip a breakpoint, enter the number of times you want to skip the breakpoint in the Number of breakpoint hits to ignore before stopping text box.
  - Click OK to commit your changes.

---

**NOTE**

The breakpoints you add, delete, or modify in this dialog box are not actually changed until you click OK.

---

## Working with deferred breakpoints

You set a deferred breakpoint when you set a breakpoint in a file or location that is part of a shared library but is not part of the current executable. This breakpoint will appear in the Breakpoints dialog list, but it will not actually be set until you load the shared library that contains the file.

### Tips

- To use pop-up menus to insert, remove, enable, and disable breakpoints:
  1. In the Source or Disassembly view, select a breakpoint or place your cursor where you want to insert a breakpoint.
  2. Right-click and click the command you want to invoke. The command will be automatically performed for the selected item without displaying the Breakpoints dialog box.
- To use the toolbar to insert and remove breakpoints:
  1. Click the toolbar button to Insert or Remove a breakpoint at the current cursor location.

## Using Breakpoints

- You can automatically set breakpoints every time the WDB GUI starts by specifying them in the `.gdbinit` file. To automatically load breakpoints, add the following line to the `.gdbinit` file:

```
break function_call
```

- You can save the breakpoints you have set by saving the debug session. You can later restore the breakpoints you have saved by restoring the debug session.

## Advancing Through Your Program

You can use the commands in this section to advance through your program during the debugging process.

You can use the Step commands to help you locate the section of your program that is causing a problem. For example, you can set a breakpoint at the beginning of a section, step through the code, and examine variables until you isolate the problem.

You can use the other commands in this section to change or show the location of the program counter.

### Stepping into functions

The Step Into command provides a single step execution of the current instruction. If the instruction is a function call, Step Into enters the function and single-steps through it.

You can use Step Into to traverse a program in detail by stepping into each instruction as it is called.

#### To step into a function or instruction

1. On the Debug menu, click Step Into.
2. Or, click the Step Into button in the toolbar:



### Stepping over functions

The Step Over command provides a single step execution of the current instruction, unless the instruction is a function call. If the instruction is a function call, Step Over executes the entire function, then pauses at the instruction immediately following the function call. Program execution also pauses at breakpoints that are set in a function.

You can use Step Over to traverse the current stack frame without stepping through the detail of each function call.

#### To step over a function or instruction

1. On the Debug menu, click Step Over.

2. Or, click the Step Over button in the toolbar:



## Stepping out of functions

The Step Out command executes the program through completion of the current function. Then the debugger pauses at the statement immediately following the function call.

You can use Step Out to return execution to the calling stack frame when you no longer want to look at the detail of a function.

### To step out of a function or instruction

1. On the Debug menu, click Step Out.
2. Or, click the Step Out button in the toolbar:



## Stepping last functions

The Step Last command steps into the function call without stepping into the argument evaluation function calls.

You can use Step Last to step into a function call without stepping through the detail of each of its argument evaluation function calls. If a function call has arguments which further makes function calls, steplast executes all argument evaluation function calls and will step into the function. Program execution also pauses at breakpoints that are set in argument function.

If steplast is not meaningful at the current line, gdb will display the following warning message: “Steplast is not meaningful for the current line; behaviour undefined.”

In C++, the steplast command is helpful while debugging heavy templated functions, because it directly steps into the call, thus skipping the constructor calls, if any. This behaviour is unlike the step command that steps into the constructor itself.

### To step last a function

1. On the Debug menu, click Step Last.

2. Or, click the Step Last button in the toolbar:



---

**NOTE** Step last feature is supported for both C and C++ languages in WDB-GUI with HP WDB version 3.2 or higher. Step last is not supported on IPF platforms.

---

## Running to the cursor

The Run To Cursor command continues executing the program until it reaches the current cursor location. The program begins execution at the program counter (indicated by a yellow arrow) and pauses at the current cursor location.

You can use the Run To Cursor command to advance to a specific location without having to set a temporary breakpoint.

### To run to the current cursor location

1. In the Source or Disassembly view, place the cursor at the location where you want program execution to pause.
2. On the Debug menu, click Run to Cursor.
  - Or, in the Source or Disassembly view, click the right mouse button, then click Run to Cursor.
  - Or, click the Run to Cursor button in the toolbar:



## Setting the next statement to execute

The Set Next Statement command sets the program counter at the current cursor location. The program begins execution from the newly set statement instead of from the previous program counter location.

You can use the Set Next Statement command to change the flow of execution if you want to re-execute or skip a portion of the program.

### To set the next statement to execute

1. In the Source or Disassembly view, place the cursor on the statement that you want to execute next.
2. Click the right mouse button, then click Set Next Statement.

### Showing the next statement to execute

The Show Next Statement command updates the WDB GUI to show the next statement to be executed. The command updates all applicable views, such as the Source, Disassembly, and Call Stack views.

You can use the Show Next Statement command to return to the current stack frame and instruction when you have been looking at a different source file or stack frame.

### To show the next statement to execute

On the Debug menu, click Show Next Statement.

- Or, in the Source or Disassembly view, click the right mouse button, then click Show Next Statement.
- Or, click the Show Next button in the toolbar:



## Setting Signal Handling

You can use Signals to view and set signal handling. This includes whether to stop for a particular type of signal, whether to let your program handle a particular type of signal, and whether to announce each type of signal.



**Table 3-2 Columns and descriptions**

Column	Description
Signal	Name of the signal
Stop	Indicates whether or not to halt execution on signal.
Print	Indicates whether or not to announce the signal

**Table 3-2 Columns and descriptions (Continued)**

<b>Column</b>	<b>Description</b>
Pass	Indicates whether or not to pass the signal to the program. Passing the signal allows the program to handle the signal.
Description	A description of the signal.

**Table 3-3 Buttons and descriptions**

<b>Button</b>	<b>Description</b>
OK	Click to apply changes you have made and exit this dialog box.
Apply	Click to apply the changes that you have made.
Cancel	Click to exit this dialog box. Changes that were not applied will be lost.

## Viewing signal handling

### To view how signals are handled

1. On the Edit menu, click Signals.
2. When you are finished, click Cancel to close the dialog box.

## Changing signal handling

### To change how a signal is handled

1. On the Edit menu, click Signals.



2. Click on the setting you want to change. Each setting toggles between Yes and No when you click on it.
3. Once you have finished changing the settings, click Apply to commit your changes.
4. Click OK or Cancel to close the dialog box.

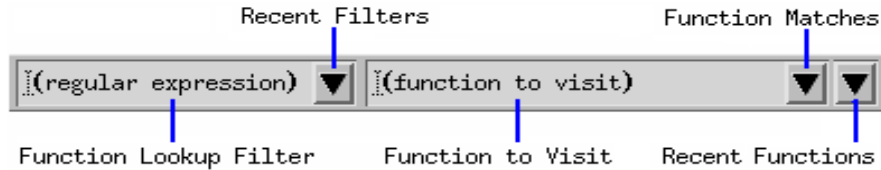
**Tips:** The signals, SIGINT and SIGTRAP, are used by the debugger. It is not a good idea to change how these signals are handled; doing so may cause the debugger to stop working.

---

## Browsing Functions

WDB GUI makes it easy to lookup and visit functions in your source code. You can specify a regular expression filter to control which functions are listed, and you can visit the source of a listed function simply by clicking on the name in the list.

The function browser is in the tool bar at the top of the debugger's main window.



### Viewing source for a function

The most direct way to view source for a function is to:

1. In the Function To Visit text box, type the function name.
2. Press RETURN. Source for the function will be displayed in the Source View.

Alternatively, if you've entered a function filter, you can select the function to view by clicking on the Function Matches button and selecting from the resulting list of functions. You can also select from a list of recently viewed functions by clicking on the Recent Functions button.

### Specifying a function filter

Specifying a function filter is useful if you don't recall the exact name of a function, or if you want to view source for a number of similarly named functions. For example, if you want to view various functions with the string "foo" in their names, do the following:

1. Type "foo" in the Function Lookup Filter text box.
2. Press RETURN. The status area at the bottom of the debugger window shows the number of matches.
3. Click the Function Matches button and choose from the list of functions. Given the filter "foo", the functions "foobar", "barfoo", and "barfoobar" would all be listed as function matches.

You can reuse a previously entered function filter by clicking on the Recent Filters button.

Function filters are regular expressions. This allows you to, for example, precede an expression with ‘^’ to match the beginning of a function name, or follow the expression with ‘\$’ to match the end of a function name.

---

**NOTE** Unless you specify otherwise, “.\*” is assumed to precede and follow the filter you enter. Examples:

- “foo” is equivalent to “.\*foo.\*”
  - “^foo” is equivalent to “^foo.\*”
  - “foo\$” is equivalent to “.\*foo\$”
- 

## Listing and viewing member functions

You can list all the member functions of a particular class by entering the class name followed by “::” in the Function Lookup Filter text box. You can then view the source for member functions by choosing the desired member function from the Function Matches list.

For example, to list function members of class Shape, you would enter “Shape::” in the Function Lookup Filter text box. You could then view the source for member functions of Shape by clicking on the Function Matches button and then selecting the member function of interest.

### Tips

- When starting a debugging session, try selecting the Recent Filters and Recent Functions buttons. Both recent function lookup filters and recently viewed functions are saved between debugging sessions.
- To specify how many filters and functions are saved across sessions, see Setting Debugger Preferences.

## Using the Program Console

The Program Console displays your program's output and accepts program input if the program uses standard UNIX I/O. Program I/O is redirected if the program performs input or output using other file descriptors or if you change the WDB GUI I/O setting.

See Setting Program Input/Output Redirection in the Loading a Program and Changing Program Settings topic for more information on standard input/output redirection.

---

**NOTE** Unless the debugger is attached to an existing process, standard output and standard error are merged in the Program Console. You cannot redirect standard error.

---

### Viewing program output

#### To open the program console

1. On the View menu, click Program Console. The Program Console is always displayed in a separate window.
2. Or, the Program Console is automatically opened whenever output is generated by the program being debugged.

### Entering program input

If your program requires input, enter it in the Program Console just as you would in a terminal window. If you enter program input while program execution is paused, the input will be buffered until program execution resumes.

#### To enter program input

1. In the Program Console, type your input.
2. Press Enter. The input is sent to your program.

---

**NOTE** If your program requires single-character input or tries to control its pty, start your program in a terminal window outside of the WDB GUI and then attach to the process. This ensures that the program you are debugging and the terminal window are connected to the same pty.

---

### Tips

- The Program Console uses a dtterm window by default. If the TERM environment variable is set to hpterm, an hpterm window is used; any other \$TERM value results in a dtterm window.
- If you close the Program Console and then reopen it, all of the text will be cleared.
- To redirect standard error output, start your program in a terminal window outside of the WDB GUI and then attach to the process. This gives you direct control of standard error output.

## Changing the Working Directory

The current working directory is the directory where the debugger runs. This is where the debugger looks for your program executable and core file. This is also one of the places the debugger looks for your source files.

The default working directory is the location where you first invoked the WDB GUI.

### To set the current working directory

1. On the File menu, click Change Directory.
2. In the Working Directory dialog box, double-click to select the appropriate directory from the Directories list.
3. When the proper directory appears in the Selection box, click OK.

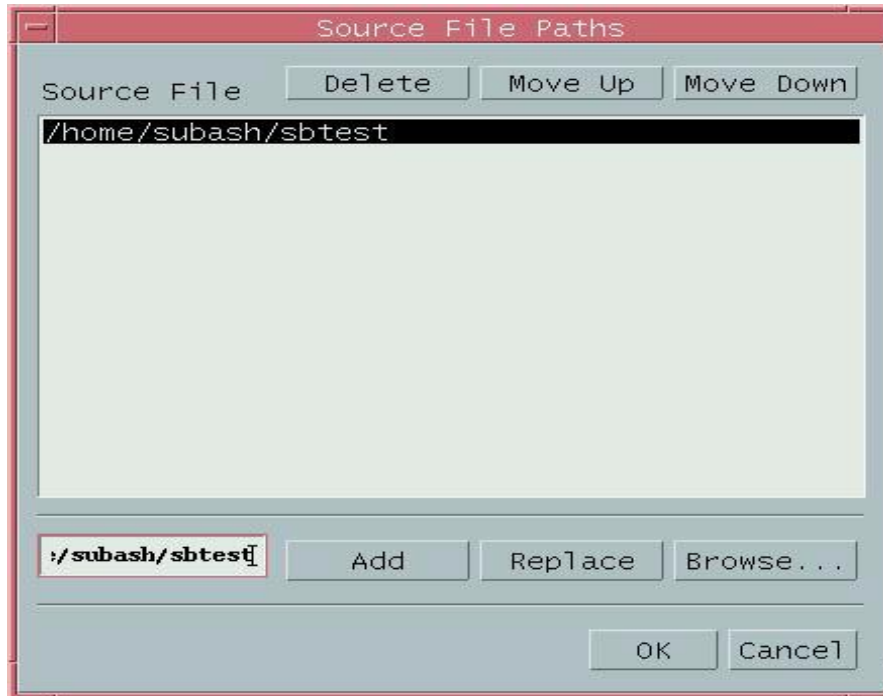
### Tips

- If your source files are not in your current working directory, you may need to add directories to your source file path.
- If your object files are not in your current working directory, you may need to add directories to your object file path.
- You can save your working directory in a debug session file. The next time you want to debug that program, you can quickly reset the working directory and other information by restoring the debug session that you saved.

---

## Setting source paths

You can set source paths so that the WDB GUI can locate the source files for the currently loaded executable. You can also delete and change the order of priority for the source paths that you have set.



## Adding and deleting source paths

### To add a source path

1. On the Edit menu, click Source File Paths.
2. Click the Browse... button in the Source File Paths dialog box.
3. In the Add Source File Path dialog box, double-click in the Directories list to select the appropriate path.

## Setting source paths

4. When the proper path appears in the Selection box, click Add.
5. In the Source File Paths dialog box, verify that the source paths you selected appear in the Source File Paths list. Click OK.

### To delete a source path

1. From the Edit menu, click Source File Paths.
2. In the Source File Paths list, select the source path you want to delete.
3. Click Delete. The selected source path disappears from the Source File Paths list.
4. Click OK.

### To replace a source path

1. On the Edit menu, click Source File Paths.
2. Double-click the directory you want to replace in the Source File Paths text box.
3. Click the Browse... button in the Source File Paths dialog box
4. In the Add Source File Path dialog box, click in the Directories list to select the new directory you want to replace the old one with.
5. When the proper directory appears in the Selection box, click OK.
6. In the Source File Paths dialog box, verify that the directory you want to replace is highlighted in the Source File Paths text box and the new directory you selected appears in the directory edit box (to the left of the Add button).
7. Click Replace.
8. Click OK.

## Changing the order of source paths

In the Source File Paths list, the paths are searched in the order they are listed. That is, the WDB GUI will search the first path listed, then the second path, and so on.

### To move a source path up

1. On the Edit menu, click Source File Paths.
2. In the Source File Paths list, select the source path you want to move up.
3. Click Move Up until the path is where you want it in the list.
4. Click OK.



### To move a source path down

1. On the Edit menu, click Source File Paths.
2. In the Source File Paths list, select the source path you want to move down.
3. Click Move Down until the path is where you want it in the list.
4. Click OK.

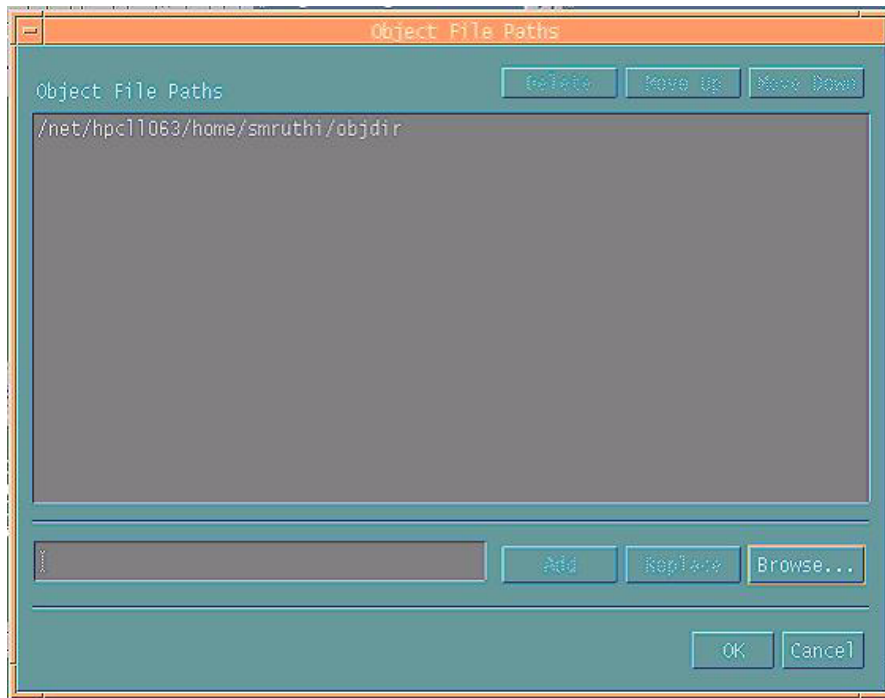
### Tips

- One way to save your source path information is to save your debug session.
- You can automatically set source file paths every time WDB starts by specifying paths in the .gdbinit file. To automatically load source file paths, add the following line to the .gdbinit file:

```
directory path
```

## Setting Object Paths

You can set object paths so that the WDB GUI can locate the object files for the currently loaded executable. You can also delete and change the order of priority for the object paths that you have set.



### Adding and deleting object paths

#### To add an object path

1. On the Edit menu, click Object File Paths.
2. Click the Browse... button in the Object File Paths dialog box.
3. In the Add Object File Path dialog box, double-click in the Directories list to select the appropriate path.

4. When the proper path appears in the Selection box, click Add.
5. Repeat steps 3 and 4 to add as many paths as you want. Click Close.
6. In the Object File Paths dialog box, verify that the object paths you selected appear in the Object File Paths list.
7. Click OK.

### **To delete an object path**

1. On the Edit menu, click Object File Paths.
2. In the Object File Paths list, select the object path you want to delete.
3. Click Delete. The selected object path disappears from the Object File Paths list.
4. Click OK.

### **To replace an object path**

1. On the Edit menu, click Object File Paths.
2. Double-click the directory you want to replace in the Object File Paths text box.
3. Click the Browse... button in the Object File Paths dialog box.
4. In the Add Object File Path dialog box, click in the Directories list to select the new directory with which you want to replace the old one
5. .When the proper directory appears in the Selection box, click OK.
6. In the Object File Paths dialog box, verify that
  - the directory you want to replace is highlighted in the Object File Paths text box.
  - the new directory you selected appears in the directory edit box (to the left of the Add button).
7. Click Replace.
8. Click OK.

### **Changing the order of object paths**

In the Object File Paths list, the paths are searched in the order they are listed, that is, the WDB GUI will search the first path listed, then the second path, and so on.

### To move an object path up

1. On the Edit menu, click Object File Paths.
2. In the Object File Paths list, select the object path you want to move up.
3. Click Move Up until the path is where you want it in the list.
4. Click OK.

### To move an object path down

1. On the Edit menu, click Object File Paths.
2. In the Object File Paths list, select the object path you want to move down.
3. Click Move Down until the path is in the place where you want it to be in the list.
4. Click OK.

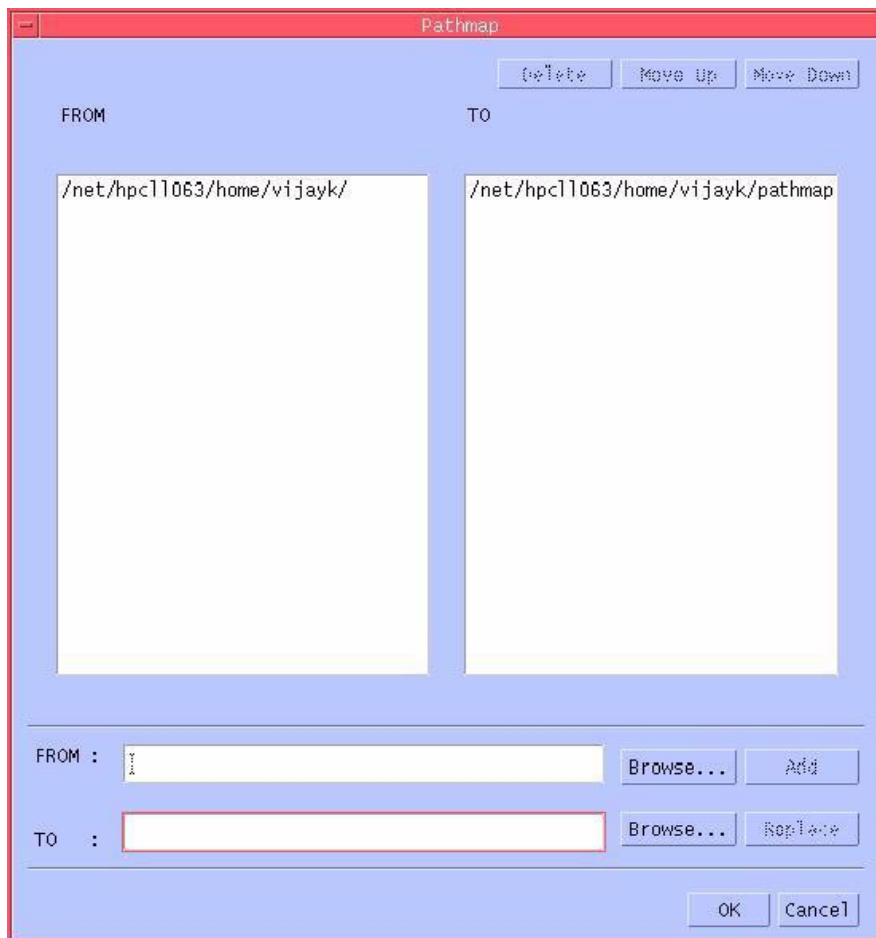
### Tips

- One way to save your object path information is to save your debug session.
- You can automatically set object file paths every time WDB starts by specifying paths in the .gdbinit file. To automatically load object file paths, add the following line to the .gdbinit file:

```
objectdir path
```

## Setting Pathmap

You can set pathmap so that WDB GUI can locate the object files for the currently loaded executable, which are spread over many directories. The Pathmap 'From' and 'To' dialog box lets you define a list of substitution rules that need to be applied to the path names to identify the object files and the corresponding source files. In this dialog box, you can add, delete, and change the priority order of pathmap.



## Adding and Deleting Pathmap

Using the Pathmap dialog box, you can add and delete pathmaps.

### To add a pathmap

1. On the Edit menu, click Pathmap.
2. In the Pathmap dialog box, click the Browse button.
3. In the From-Add Pathmap File Path and To-Add Pathmap File Path dialog box, double-click in the Directories list to select the appropriate 'From' and 'To' paths.
4. When the proper path appears in the Selection box in the From-Add Pathmap File Path and the To-Add Pathmap File Path dialog boxes, click Add to add in Pathmaps in the From and To text boxes in the Pathmap dialog box.
5. Click Add in Pathmap dialog to add pathmaps in pathmap list.
6. In the Pathmap dialog box, verify that the pathmaps you selected appear in the Pathmap list.
7. Click OK.

### To delete a pathmap

1. On the Edit menu, click Pathmap.
2. In the Pathmap list, select the pathmap that you want to delete.
3. Click Delete. The selected Pathmap disappears from the Pathmap list.
4. Click OK.

### To replace a pathmap

1. On the Edit menu, click Pathmap.
2. Double-click the directory you want to replace in the Pathmap text box, it will be listed in 'From' and 'To' path selection text boxes..
3. Click Browse button, click in the Directories list in the select From-Add Pathmap File paths and To-Add Pathmap File paths to select the new directory with which you want to replace the old one .
4. When the proper directory appears in the Selection box, click OK.
5. In the Pathmap dialog box, verify whether:

- the directory that you want to replace is highlighted in the Pathmap- From and To text box.
  - the new directory that you selected appears in the directory edit box (to the left of the Browse button).
6. Click Replace.
  7. Click OK.

## Changing the Order of Pathmap

In the Pathmap list, the paths are searched in the order in which they are listed, that is, the WDB GUI will map the first pathmap listed, then the second path, and so on.

### To move a pathmap up

1. On the Edit menu, click Pathmap.
2. In the Pathmaps list, select the pathmap you want to move up.
3. Click Move Up until the path is where you want it in the list.
4. Click OK.

### To move a pathmap down

1. On the Edit menu, click Pathmap.
2. In the Pathmap list, select the pathmap you want to move down.
3. Click Move Down until the path is in the place where you want it to be in the list.
4. Click OK.

### Tip

One way to save your pathmap information is to save your debug session.

## Opening Source Files

You can open and view a source file associated with the executable that you are debugging. For example, you might want to set a breakpoint in a particular source file that is associated with the current executable, or you might want to edit and rebuild a file in the debugger.

### To open a source file

1. On the File menu, click Open File.
2. In the Directories list, double-click to select the appropriate directory.
3. Select the Open in debugger toggle (this is the default). You have the option of opening a file in an external editor. This is useful when you want to edit a file in an external editor, but you also want the debugger to monitor the file for changes, and rebuild the file if changes are detected. See Fixing Code from within the Debugger for details.
4. In the Files list, select the appropriate file, then click OK. The file will appear in its own editable tabbed window in the source area. In the tabbed window, you can set breakpoints just as you would in the Source view. You can also edit the file.

### To reopen a source file that was recently open

1. On the File menu, click Recent Files. A menu of recently open files will appear.
2. Click the name of the file you want to open.

### To save source files

You can save a source file that you've edited in the debugger using any of the following methods:

- Enter Ctrl+S.
- Click on the Save File icon in the toolbar:



- Choose Save File from the edit window pop-up menu (press M3).
- Choose Save File from the File menu.




## To close a source file

You can close a source file using one of the following methods:

1. Choose Close File from the edit window pop-up menu (press M3).
2. Choose Close File from the File menu.

### Tips

- Click the  toolbar button to open the Open File dialog box:



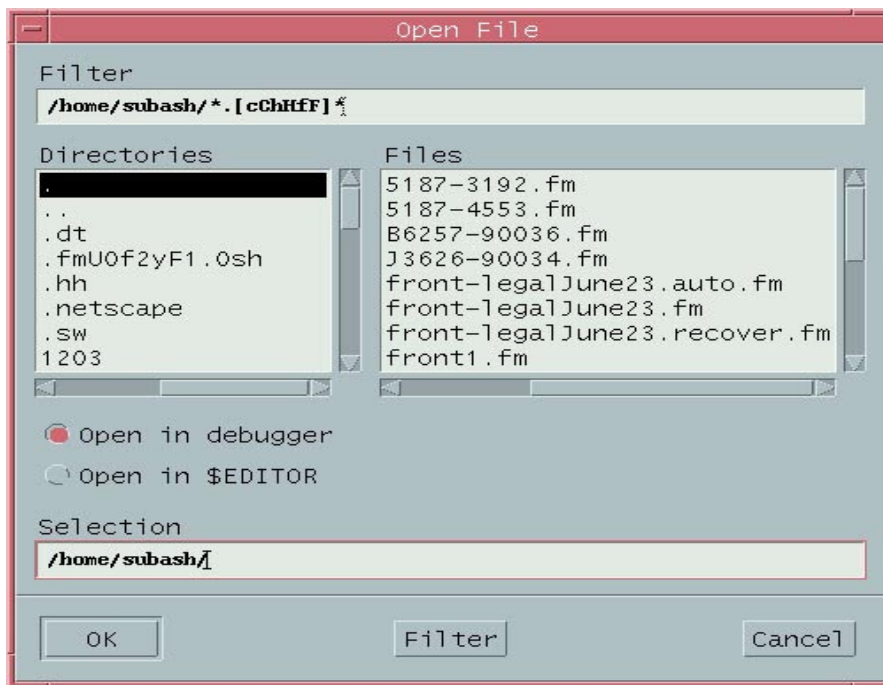
- To change the size of the Recent Files list on the File menu, click Preferences on the Edit menu, and then change the value of Recent files list, maximum size.
- You can open another tab for a file in the Source view by choosing Edit File from the Source view pop-up menu (press M3).
- You can open a file in the Source view in an external editor by choosing Open File in \$EDITOR from the Source view pop-up menu (press M3). If the EDITOR environment variable is undefined, vi is used.
- You can open a file associated with a particular function using the function browser (see Browsing Functions for details).

---

## Viewing Code

The WDB GUI offers two main views for looking at your code: the Source view and the Disassembly view. Use the Source view to see the contents of a file, and use the Disassembly view to see the corresponding assembly code.

The dwell feature allows you to quickly view the value of a variable or expression. It is available from both the Source view and Disassembly view.



## Viewing source code

Before you can view your source code, you must have an executable loaded and running in the WDB GUI. Once you have loaded a file, you can use the Source view to see and manipulate your source code. Specifically, you can use the Source view to:

- See the current file's source path.
- See the source file contents.

- See the current location of the program counter. The program counter is set to the point in your program where the debugger will begin executing.
- See the location and status of breakpoints.
- See a stack marker that indicates the current context, determined by the stack frame selected in the Call Stack view.
- Access the pop-up menu to perform actions that will affect how the code executes.
- Use dwell to quickly view the value of a variable or expression.

### **To view your source code in the source view**

1. In the main window, click the Source tab.
2. View the file's source path by looking below the Source tab label.
3. View the source code content in the right column of the Source view.
4. View the location of the program counter, breakpoints, and the stack marker in the left column of the Source view.

---

**NOTE** You cannot display the Source view in a separate window.

---

### **To use the pop-up menu in the Source View**

In the Source view, click the right mouse button to display a pop-up menu that contains the following commands:

- Show Next Statement
- Quick Watch
- Insert/Remove Breakpoint
- Enable/Disable Breakpoint
- Edit Breakpoint
- Run to Cursor
- Set Next Statement
- Edit File
- Open File in \$EDITOR

## Viewing assembly code

Before you can use the Disassembly view, your executable must be running. You can use the Disassembly view to:

- See the assembly code associated with the currently executing function.
- See the instruction that corresponds to the current location of the program counter. The program counter is set to the point in your program where the debugger will begin executing.
- See the location and status of breakpoints.
- See a stack marker that indicates the current context, determined by the stack frame selected in the Call Stack view.
- Access the pop-up menu to perform actions that will affect how the code executes.
- Perform debug actions, such as stepping through a program and setting breakpoints.
- Use dwell to quickly view the value of a variable, register, or expression.

### To view code in the Disassembly View

1. In the main window, click the Disassembly tab.
2. View the assembly code content in the right column of the Disassembly view.
3. View the location of the program counter, breakpoints, and the stack marker in the left column of the Disassembly view.

### To see the Disassembly View in a separate window

- Click the right mouse button, then click Open Separate View. The separate window behaves the same as the tabbed Disassembly view.
- To close the separate window, click the right mouse button, then click Close.

### To use the pop-up menu in the Disassembly View

In the Disassembly view, click the right mouse button to display a pop-up menu that contains the following commands:

- Show Next Statement
- Quick Watch
- Insert/Remove Breakpoint
- Enable/Disable Breakpoint

- Edit Breakpoint...
- Run to Cursor
- Set Next Statement
- Open Separate View
- Close

## Viewing variables using Dwell

The dwell feature displays the value of the variable or expression on which the mouse pointer is paused. Dwell works in both the Source view and the Disassembly view.

Dwell gives you quick information about a variable. For more detailed information, use Quick Watch, Watch view, or Local Variables view.

### To view the value of a variable or expression

1. Use the mouse to position the pointer over a variable. For an expression (such as a+b), highlight the expression to select it, then point to the highlighted text.
2. Do not move the pointer. After a short time (about a half-second), a window appears that displays the value of the variable or expression.
3. Move the mouse or press a key or button to close the dwell window.
4. You can view the decimal and hexadecimal value of a numeric variable or expression by changing the preference in the Source View section of the Edit > Preferences dialog box. See Setting Debugger Preferences. The displayed decimal and hexadecimal values are separated by a semicolon(;)

### Limitations of Dwell

Dwell does not display the value of certain types of variables and expressions. It will not display the value of complex variables such as arrays and structs, but it will display the value of elements in an array, fields in a struct, and pointers.

Expressions are not displayed if there may be a potential side effect on the program being debugged (such as i++ or exit(1)). Expressions containing ), =, ++, and -- are not displayed with dwell.

### Examples of complex variables

Dwell displays the value of fields as well as variables. Dwelling on bar of foo->bar displays the value of foo->bar, while dwelling on foo displays only the value of foo.

In the expression a[i] there are several places to use dwell:

## Viewing Code

- If `a` is a pointer, dwelling on `a` displays the value of variable `a`.
- If `a` is an array, dwelling on `a` displays the text `[...]`. This indicates that the variable is an array and cannot be viewed with `dw`. You can use `dw` to view the value of individual elements within an array.
- Dwelling on `i` displays the value of `i`.
- Dwelling on `[i]` displays the value of `a[i]`.

In the expression `*p`, you can use `dw` in the following ways:

- Dwelling on the `*` displays the value of `*p`
- Dwelling on the `p` displays the value of `p`

### Tips

- Use the function browser to view source for a particular function. See [Browsing Functions](#) for details.
- Click the `Disassembly` toolbar button to open and close the Disassembly view in a separate window.
- On the `View` menu, click `Source` or `Disassembly` to quickly bring that view to the top.
- You can customize the WDB GUI to display line numbers in the Source view. On the `Edit` menu, click `Preferences` and use the `Display Line Numbers` check box to set whether line numbers are displayed in the Source view.

---

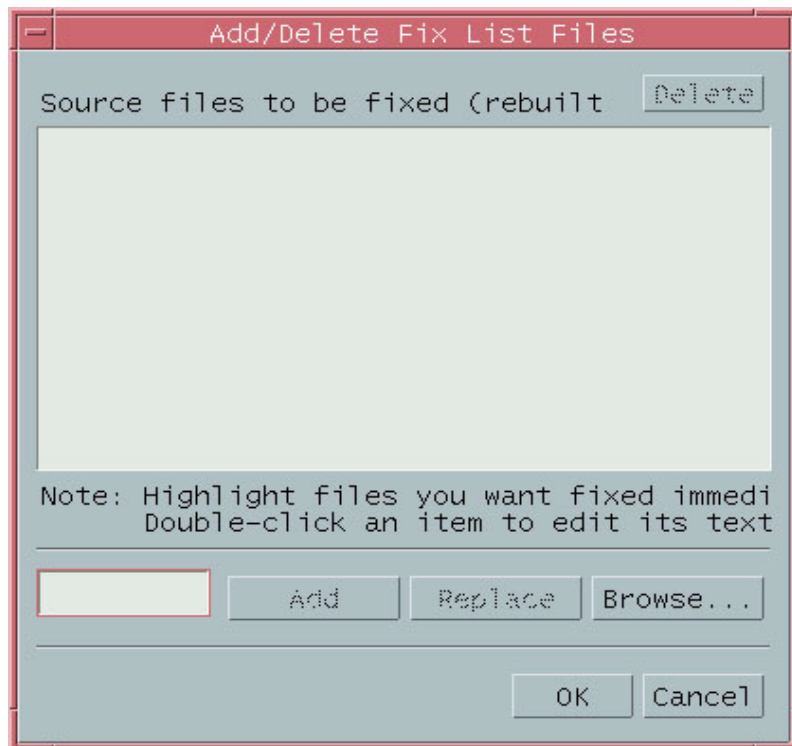
## Fixing the Code from within the Debugger

HP WDB lets you change the program you are debugging without having to exit the debugger or even restart the executable you are debugging. You have the option of explicitly requesting that a file be rebuilt and patched in, or you can have the debugger monitor source files, rebuilding the executable whenever changes are detected.

---

**NOTE** Changes made to an executable during a debugging session are not saved when you exit the debugger. To apply the changes outside the debugger, you must rebuild your executable. There are a number of restrictions on the changes you can make to your program while it is being debugged. See Restrictions on Fixing Code from within the Debugger for details.

---



## Fixing code and continuing your program

### To edit your source code in the debugger

1. On the File menu, click Open File.
2. In the Directories list, double-click to select the appropriate directory.
3. Select the Open in debugger toggle (this is the default). If you prefer, you can choose to open the file in an external editor.
4. Make changes to the source file or files.
5. If you're using an external editor, save your changes when you have completed them.
6. Click on one of the following toolbar buttons:
  - Step Into
  - Step Over
  - Step Out
  - Step Last
  - Go
  - Restart
7. WDB GUI asks you if you want to apply the source changes. Click Yes. The debugger performs the command you selected using the modified source.

The Build View tab displays any build output and reports when the build has completed.

---

### NOTE

- When the debugger starts an external editor, it chooses the editor based on the environment variable EDITOR. If EDITOR is not set, the debugger starts vi.
  - Files you open are monitored for changes (text changes for the debugger editor; modification times for external editors). When you click on a program execution button (such as Step Into), the debugger asks you if you want to rebuild changed files.
  - The fix and continue feature is not supported on IPF platform.
-



## Explicitly requesting the debugger for monitoring or rebuilding files

You can explicitly request that the debugger monitor or fix a specific file or files. This is useful, for example, for building dependent files that need to be rebuilt because you change a header file.

### To explicitly request for monitoring a specific file for changes

1. On the Edit menu, click Fix List.... The Add/Delete Fix List Files dialog appears.
2. Click Browse... to display the Name of File to Fix dialog box.
3. In the dialog box, choose a file to add to the fix list.
4. Click OK to close the dialog box. The file you selected appears in the Add/Delete Fix List Files dialog.
5. When you are finished adding files to the fix list, Click OK. All files in the list will be monitored for changes.

### To request an immediate rebuild of a specific file

1. On the Edit menu, click Fix List.... The Add/Delete Fix List Files dialog box appears.
2. In the Fix List dialog, highlight any of the files in the fix list that you want rebuilt and patched in immediately.
3. Click OK in the Fix List dialog. The files you highlighted will be rebuilt and patched in immediately.

## Examining build errors and warnings

Build errors and warnings are posted to the Build View. The debugger displays the Build View automatically when the debugger rebuilds code you have fixed.

To view the source code for an error or warning, double click on the error or warning in the Build View. The source associated with the error or warning will be displayed in the Source View.

## Restrictions on fixing code from within the debugger

These types of source code changes cannot be fixed within the debugger:

- Adding, deleting or reordering the local variables and parameters in a function currently active on the stack.

- Changing the data type of a local variable, file static, global variable, parameter of a function.
- Changing a local, file static, or global variable to be a register variable, or vice-versa.
- Adding any function calls that increase the size of the parameter area.
- Adding an `alloca()` function to a frame that did not previously use `alloca()`.
- Adding fields to a structure anywhere other than at the end of the structure.
- Changing the order of fields in a structure.

Additionally, the debugger has these limitations in how it handles changed code:

- With the exception of the routine at the top of the stack, new code will not be executed for a routine currently on the stack until that routine is popped off the stack and program execution reenters that routine.
- With the exception of the routine at the top of the stack, breakpoints in a routine on the stack will be inactive until that routine is popped off the stack and program execution reenters that routine.
- If the name of a function is changed and there was a breakpoint set to the old version of the function, the breakpoint is not moved to the new function. Additionally, the old breakpoint is still valid (and can be hit when existing routines call the old function name).
- If the number of lines of the modified file is different from that of the original file, the placement of breakpoints may not be correct.
- When the program resumes, the program counter is moved to the beginning of the same line in the modified function. It is possible that the program counter may be at the wrong line.

---

## Finding Specific Text in Your Code

You can use Find and Find Next to search your current view for specific text or a regular expression. It is available in the following views: Source, Disassembly, Command, Call Stack, and Threads. A blue line highlights the current view.

You can choose to search for matching upper/lower case and to direct the search toward the beginning or end of the file. The cursor is moved to the next match in the file, and the matching text is highlighted.

Find Next searches for the next occurrence of the text or regular expression specified the last time you used Find.



### To find text or a regular expression

1. On the Edit menu, click Find.
2. In the Find box, type the text or regular expression that you want to find. If you use regular expressions, ensure that the Regular Expression check box is selected.

Unless the Regular Expression check box is selected, the text you enter in the Find box is matched literally. For example, wildcard characters are not supported.

3. Select the settings for your search:
  - Match Case - If this option is checked, the case of each letter must match; otherwise a match can occur regardless of case.
  - Regular Expression - If this option is checked, the text in the Find box is interpreted as a regular expression for matching a pattern.

- Direction - The Up or Down options direct the search toward the beginning or end of the source file. The search begins at the current cursor location and wraps around to the opposite end of the document.
4. Click Find Next to begin the search. The cursor is moved to the next match in the file, and the matching text is highlighted.
  5. When you are finished, click Cancel to close the dialog box.

### To find the next match

On the Edit menu, click Find Next. The cursor is moved to the next match in the file, and the matching text is highlighted.

### Tips

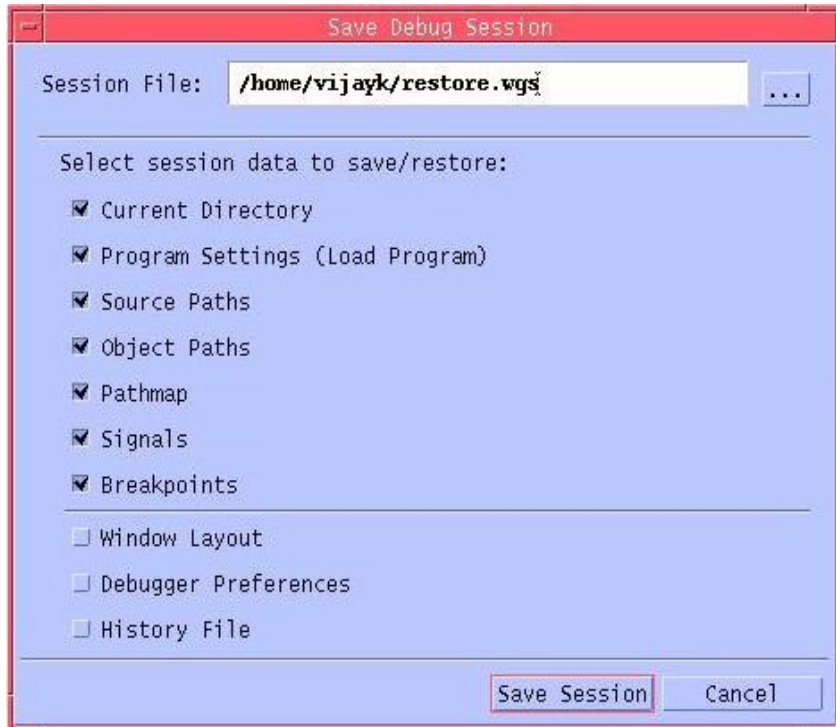
- Click the toolbar button to open the Find dialog box:



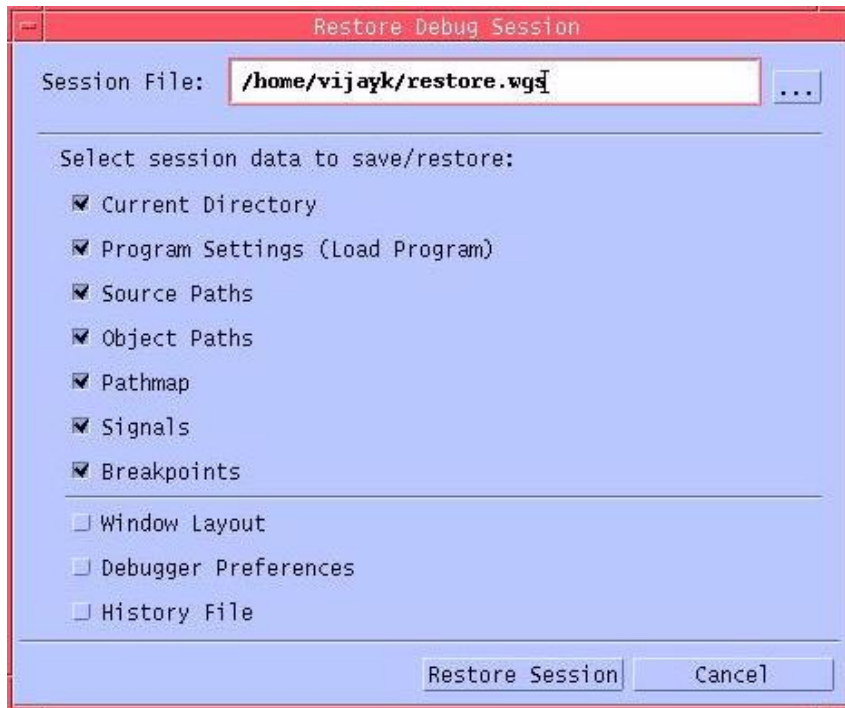
- The last text string for which you searched is entered in the Find text box.
- To find text in a specific view, click in that view.

---

## Saving and Restoring Debug Sessions



The Save Debug Session dialog box



The Restore Debug Session dialog box

You can save and restore the following debug session attributes:

- Current working directory
- Program settings
- Source paths
- Object paths
- Pathmap
- How signals are handled
- Current breakpoints
- Window layout
- Debugger preferences

## To save a debug session

1. On the File menu, click Save Session.
2. In the Save Debug Session dialog box, enter the name of the file where you want the session to be saved. You can type or browse for the file name.
3. Check the types of session data you want to save.
4. Click OK.

## Restoring a debug session

1. On the File menu, click Restore Session.
2. In the Save Restore Session dialog box, enter the name of the session file you want to restore. You can type or browse for the file name.
3. Check the types of session data you want to restore.
4. Click OK.

---

**NOTE** Restoring the current directory or program settings will cause your program to reload.

---

### Tips

- A quick way to restore a debug session is to use the Recent Sessions list on the File menu. This restores all the session data that was saved in the file you select.
- When you restore a session, the available session data will be checked automatically.
- Restoring the current directory or program settings will cause your program to reload. You may want to uncheck these if you do not need to restore this information.
- If you save a session to a file that already has session data in it, the types of session data you chose to save previously will be checked by default.





---

## **4 Troubleshooting WDB GUI**

## Error Messages

The following table explains the causes and solutions for the various error messages that the system displays while you work with WDB GUI:

**Table 4-1 Error Message descriptions**

Message	Cause	Solution
Failed to find debugger	<p>1.If gdb is not in the default path /opt/langtools/bin/</p> <p>2. If GDB_SERVER is not set to a valid path</p>	<p>1. Export GDB_SERVER=&lt;gdb_path&gt;</p> <p>2. Export GDB_SERVER to valid path (pointing to gdb). export GDB_SERVER=/path/to/gdb</p>
Cannot determine location for current line.	<p>The cursor is in an invalid location in the Source view when you initiate one of the following operations:</p> <ul style="list-style-type: none"> <li>• Insert/Remove Breakpoint</li> <li>• Run To Cursor</li> <li>• Set Next Statement</li> </ul>	<p>Move the cursor to a valid source line in the Source view.</p>
Failed to change to directory	<p>You have entered an invalid path for the current working directory.</p>	<p>From the Load Program dialog box, click the Run tab, and check the Working Directory text entry.</p>
No ptys available.	<p>You have tried to open a program console terminal and all ptys are in use. Or you are denied access to the pty.</p>	<p>Close some applications that use ptys, then shut down and restart the WDB GUI. Or check pty ownership then try again.</p>

**Table 4-1 Error Message descriptions (Continued)**

<b>Message</b>	<b>Cause</b>	<b>Solution</b>
Unable to make request -- gdb not connected	<p>The gdb backend is not connected to the WDB GUI because:</p> <ul style="list-style-type: none"> <li>• The backend is in the process of connecting to the WDB GUI.</li> <li>• Or the backend has unexpectedly gone away</li> </ul>	<p>Wait until the backend is finished connecting to the WDB GUI.</p> <p>Or restart the WDB GUI.</p>
Unable to open slave pty.	You have tried to open a program console terminal and you do not have proper permissions for the slave pty.	Close some applications that use ptys, then shut down and restart the WDB GUI. Or check pty ownership then try again.
Unable to start terminal.	You have tried to open a program console terminal and the terminal executable is not available. Or, too many processes are running.	Make sure that the terminal executable is available, then try again. Or, close some applications to kill processes, then shut down and restart the WDB GUI. Or, try setting your TERM environment variable to hpterm or dtterm.
Underlying gdb exited.	The gdb debugger backend process exited or was killed.	Shut down and restart the WDB GUI.
usage: wdb [gdb options] [Xt options] [-session filename] [executable-file [core-file or process-id]].	You have entered invalid command options.	Check command options based on the usage line provided in this error message, then try again.

**Table 4-1 Error Message descriptions (Continued)**

<b>Message</b>	<b>Cause</b>	<b>Solution</b>
Unrecognized file format.	The Session or Preferences file cannot be parsed. The file is probably not a session file or has been overwritten by some other application.	Choose a different session file. If you see this during startup, delete the Preferences file, \$HOME/.wdbguirc.
Wrong version of the file format.	The Session or Preferences file has a format that cannot be parsed.	Choose a different session file. If you see this during startup, delete the Preferences file, \$HOME/.wdbguirc.
Error: No session file name provided: -session	No session file name was provided with the -session command line argument.	Provide a session file name, or do not use the -session option.
File not found: index.html	When you clicked Topics on the Help menu, the WDB GUI could not find the online help file.	Install the online help fileset for the WDB-GUI product.
File not found: feedback.html	When you clicked Feedback on the Help menu, the WDB GUI could not find the online help file.	Install the online help fileset for the WDB-GUI product.

---

**NOTE** See the Command view output area for additional debug messages and warnings.

---

## Troubleshooting Tips

### Debugging executables over NFS

You can debug executables that are stored on NFS-mounted file systems. However, you cannot attach to an existing process if the executable you are running resides on an NFS-mounted file system instead of on your local machine's file system. To work around this problem, copy the executable onto your local machine's file system, restart the process on your local machine, run the debugger, and attach to the process.

### Attaching to a process with shared libraries

If you want to attach to a running process that contains shared libraries, before you run your program, you need to run the command `/usr/bin/pxdb -s enable executable_file` where `executable_file` is the name of your program executable.

### Double-Clicking with ReflectionX

If you are running the WDB GUI using ReflectionX, the default double-click parameters may be too sensitive to work properly. You can increase the double-click time by setting the following X resource:

```
Wdb*multiClickTime: 600
```

For information on setting X resources, refer to the `xrdb(1)` man page or to the RESOURCES section of the `x(1)` man page.

### Setting breakpoints that never appear in the breakpoints list

You may set a breakpoint that does not show up in the breakpoints list. This may be caused by:

- The breakpoint has been deferred. See Working with Deferred Breakpoints.
- Or a breakpoint was actually set as you specified, however the debugger cannot locate the appropriate file using the current source file paths that you have set. To update the source file paths, see Setting Source Paths.

## **Problems with keyboard shortcuts or mneumonics**

If you have problems with keyboard shortcuts or mneumonics, check if the CAPS LOCK or NUM LOCK key is active. These modifier keys disable most shortcut and mnemonic features. To use the short cuts and mneumonics, be sure that these two modifier keys are de-activated.

## Using Online Documentation

### Setting up a Web browser to view Help

When you click Help Topics or Feedback on the Help menu, the WDB GUI attempts to invoke a local web browser. If the default command fails, the WDB GUI will issue a warning and allow you to edit the browser command.

#### To change the browser command

1. Click Preferences on the Edit menu.
2. Click the Help tab.
3. Edit the browser command in the Browser Command text box.
4. Click OK.

The new browser command is stored in the file `$HOME/.wdbguirc` when the WDB GUI exits normally.

### Locating information

#### Table of Contents

To find information, browse the Table of Contents in the left column of each page. It lists all of the documents available in WDB GUI Help. When you click on a document, it expands to list all of the topics it contains.

Click on a topic to display the following types of information on the right side of the page:

- Topic contents -- Provides links to all of the sections in the topic.
- Overview -- Offers a summary of the topic and other general information.
- Task or reference information -- Provides the steps to complete a task or describes a view, window, or dialog box in the WDB GUI.
- Tips -- Highlights tips, shortcuts, and other special information about the topic.
- See Also -- Provides links to related topics.

- Keyword Index

You can use the index to look up specific terms. Each term is followed by a list of topics containing the term or concept. Click on a topic to display its contents.

## Understanding the navigation icons

The following icons are at the top of each page to help you move around WDB GUI Help:

Returns you to the WDB GUI Help home page.

Takes you to Using Help, the topic you are currently viewing.

Takes you to the WDB GUI keyword index.

## Printing Topics

You can print any HTML topic using the Print command in your Web browser.



## Getting Support

If you have a support contract, call the HP Response Center at 800-633-3600 (North America) or contact your Country Response Center. You will need the software system identifier (system handle) associated with your support contract. HP WDB is covered under the support contract for your HP compiler.

For inquiries about your system handle or support contract, call 800-386-1115.

If you have a problem for which a defect needs to be reported, provide a test case demonstrating the problem. The Response Center Engineer will then enter a defect report.

For the latest HP WDB debugger information, visit:

<http://www.hp.com/go/wdb>



Home



Help



Index