

HP WDB GUI Reference Manual

HP-UX 11i v1, HP-UX 11iv2, and HP-UX 11i v3

Edition 11



i n v e n t

Manufacturing Part Number: 5992-0595

February 2008

Printed in the United States

© Copyright 2008 Hewlett-Packard Development Company, L.P.

Legal Notices

© Copyright 2008 Hewlett-Packard Development Company, L.P.

Confidential 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.

The information contained herein is subject to change without notice. The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

UNIX is a registered trademark of The Open Group. Intel and Itanium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

1. WDB GUI Overview	
Summary of Features	11
2. Installing WDB GUI	
Installation Overview	20
Supported Configurations	20
Disk Space Requirements	20
Installation Instructions	20
Filesets	20
3. Debugging with WDB GUI	
Loading a Program and Changing Program Settings	24
Setting Run Options	25
To specify an executable:	25
To load a core file for debugging	25
To attach to an existing process for debugging	25
To detach process	26
Setting Environment Variables	26
To view current environment variables	27
To set new environment variables	28
To delete environment variables	28
Setting Program Input/Output Redirection	28
To redirect standard input and output information	29
Starting and Stopping the Debug Process	30
Starting the debug process (Go)	30
To start debugging	30
Restarting the debug process (Restart)	30
To restart the debug process	31
Stopping a debug session	31
To stop a debug session	31
Pausing program execution	31
To pause program execution	31
Using Breakpoints	32
Tip	33
Inserting breakpoints	34
To insert a breakpoint	34

Removing breakpoints	34
Viewing and modifying breakpoints	35
To bring up the Breakpoints dialog box	35
To view attributes of a breakpoint	35
To modify attributes of a breakpoint	35
Working with deferred breakpoints	36
Tips.	36
Advancing Through Your Program	37
Stepping into functions.	37
To step into a function or instruction	37
Stepping over functions	38
To step over a function or instruction	38
Stepping out of functions	38
To step out of a function or instruction	38
Stepping last functions.	38
To step last a function.	39
Running to the cursor.	39
To run to the current cursor location	39
Setting the next statement to execute.	40
To set the next statement to execute	40
Showing the next statement to execute.	40
To show the next statement to execute	40
Setting Signal Handling	40
Viewing signal handling.	42
To view how signals are handled	42
Changing signal handling	42
To change how a signal is handled.	42
Browsing Functions	43
Viewing source for a function.	43
Specifying a function filter.	43
Listing and viewing member functions	44
Tips.	44
Using the Program Console.	44
Viewing program output.	45
To open the program console	45
Entering program input	45

To enter program input	45
Changing the Working Directory	46
To set the current working directory	46
Tips.	46
Setting source paths.	47
Adding and deleting source paths	47
To add a source path	47
To delete a source path	48
To replace a source path	48
Changing the order of source paths	48
To move a source path up	48
To move a source path down.	49
Setting Object Paths	49
Adding and deleting object paths.	50
To add an object path	50
To delete an object path	51
To replace an object path	51
Changing the order of object paths	51
To move an object path up	51
To move an object path down	52
Setting Pathmap	53
Adding and Deleting Pathmap.	54
To add a pathmap	54
To delete a pathmap	54
To replace a pathmap	54
Changing the Order of Pathmap	55
To move a pathmap up	55
To move a pathmap down	55
Tip	55
Opening Source Files	55
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	56
Tips.	57
Viewing Code	57

Viewing source code	58
To view your source code in the source view	59
To use the pop-up menu in the Source View	59
Viewing assembly code	59
To view code in the Disassembly View	60
To see the Disassembly View in a separate window	60
To use the pop-up menu in the Disassembly View	60
Viewing variables using Dwell	61
To view the value of a variable or expression	61
Fixing the Code from within the Debugger	62
Fixing code and continuing your program	63
To edit your source code in the debugger	63
Explicitly requesting the debugger for monitoring or rebuilding files	64
To explicitly request for monitoring a specific file for changes	64
To request an immediate rebuild of a specific file	65
Examining build errors and warnings	65
Restrictions on fixing code from within the debugger	65
Finding Specific Text in Your Code	66
To find text or a regular expression	67
To find the next match	68
Tips	68
Displaying Memory Layout	69
Viewing Assembly Instruction Description	70
Saving and Restoring Debug Sessions	71
To save a debug session	73
Restoring a debug session	73
Tips	73
Debugging Memory Problems	74
Viewing Memory Leaks and Heap Usage	76
Viewing the Runtime Heap and Leak data graphically	76
Viewing the Incremental Heap Profile	76
Viewing the Batch RTC Report	77
Viewing the Arena Profile	78
Enabling and Disabling Specific Threads	81
Advanced Thread Debugging Support	82
Support for info thread, info thread <thread-id> and set thread check <on/off> . . .	82

Support for info mutex and info mutex <mutex-id>	83
Support for info rwlock and info rwlock <rwlock-id>	83
Support for info condvar and info condvar <condvar-id>	84
Debugging Inline Functions	85
Using Inline Debugging Options	85
Viewing the Execution Path Entries	87
Compiler Dependencies	87
Displaying the Execution Path Entries	87

4. Troubleshooting WDB GUI

Error Messages	92
Troubleshooting Tips	95
Debugging executables over NFS	95
Attaching to a process with shared libraries	95
Double-Clicking with ReflectionX	95
Setting breakpoints that never appear in the breakpoints list	95
Problems with keyboard shortcuts or mnemonics	96
Known Problems	97
Using Online Documentation	100
Setting up a Web browser to view Help	100
To change the browser command	100
Locating information	101
Table of Contents	101
Understanding the navigation icons	101
Printing Topics	102
Getting Support	102

1 WDB GUI Overview

The HP WDB GUI is a Graphical User Interface (GUI) designed by Hewlett-Packard for HP WDB. HP WDB (Wildebeest Debugger) is the HP implementation of the open source debugger GDB. It supports the debugging of 32-bit and 64-bit programs written in HP C, HP aC++, and Fortran 90 on Integrity systems running HP-UX 11i v2 or HP-UX 11i v3, and HP 9000

systems running HP-UX 11i v1, HP-UX 11i v2, or HP-UX 11i v3.

Using WDB GUI, you can do the following:

- Debug an executable file or a core file produced by an aborted program
- Attach to a running process
- 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 leak report
- Edit sources and compile the changed source without restarting the program being debugged
- View the memory layout
- View the shared libraries
- View information on `pthread` primitives
- Debug inline functions
- View the arena profile
- View the execution path entries
- View graphical display of Batch RTC Reports
- View graphical display of runtime heap and leaks

You can 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 graphical display of batch RTC reports**

HP WDB GUI 5.8 and later versions provide graphical representation of batch RTC reports. HP WDB GUI displays:

- Batch RTC reports on the basis of function names instead of block or byte
- Leak/Heap/Corruption reports
- Data for Heaps or Leak in the form of Pie Charts
- Corruption Reports in tabular form
- Data which you can sort on the basis of No.of Bytes, Blocks or Callee

- **Support for graphical display of runtime Heap and Leaks**

HP WDB GUI 5.8 and later versions provide graphical representation of runtime Heap and Leaks. HP WDB GUI displays:

- Runtime heap or leak data on the basis of function names
- Data in the form of Pie Charts which you can sort on the basis of No.of Bytes, Block, or Callee

- **Support for display of Nat Values for Floating Point Registers**

HP WDB GUI 5.8 and later versions provide support for display of Nat Values for Floating Point Registers, if any.

- **Support for different display formats in Register View**

HP WDB GUI 5.8 and later versions provide support for different display formats for integer registers. The available formats are Decimal, Octal and Hexadecimal.

- **Support for enhanced display of Execution Path**

HP WDB GUI 5.8 and later versions displays many source and assembly instructions in respective views or tabs.

- **Support for Viewing the Incremental Heap Profile**

HP WDB GUI 5.7 and later versions provide support to view the incremental heap profile of an application.

- **Support for the Mouse Scroll Wheel**

- **Support for Viewing the Execution Path Entries**

HP WDB GUI 5.7 and later versions provide support to view the execution path for programs running on Integrity systems. This feature is supported only for compiler versions A.06.15 and later.

- **Support for viewing information on the state of pthread primitives**

HP WDB GUI 5.6 and later, provide support to view extended information on the state of threads, mutexes, read-write locks, and condition variables.

- **Support for heap check string on/off**

WDB GUI supports the heap-check string on/off feature of GDB. If the check string flag is set to on, it enables the validation of calls to functions such as `strcpy`, `strncpy`, `memcpy`, `memncpy`, `memmove`, `memset`, `bzero`, and `bcopy`.

- **Support for display of memory layout for a given address**

WDB-GUI displays memory contents for a given address, spanning across 2 kB of memory. This option is available from the View menu. The contents can be viewed in decimal, hexadecimal, octal, unsigned decimal, binary, float, address, instruction, character and, string format. For a given display format, the contents can be viewed in byte, halfword and word sizes.

- **Support for display of a short description for an assembly instruction**

WDB-GUI displays a brief description of the assembly instruction when the mouse pointer is moved over an assembly instruction in the Disassembly view.

- **Special key strokes for minimizing and maximizing WDB GUI windows**

WDB-GUI window can be minimized and maximized `Control + Shift + <` and `Control + Shift + >`.

- **Support for shared library status**

This feature lists all the libraries loaded by a program that is being debugged.

- **Support for display of current program information**

- **Improved browser selection screen for viewing HP WDB GUI Online Help**

The browser preferences screen has been modified. The user can select either Netscape or Mozilla as the browser for viewing the HP WDB GUI Online Help. Toggle buttons have been placed on the browser selection screen for the two browsers supported. If the default browser (in this case Netscape) is not installed on the machine, then the user is prompted to select an alternative browser on the browser selection screen.

- **Changed registers displayed in red**

The register view of the HP WDB GUI, is updated everytime the debugged program stops due to some event.

- **Support for viewing array contents added to HP WDB GUI**

The user can view the values for part of an array by defining the limits. The array contents are displayed for specific limits only. However, by default the entire array is displayed.

- **Support for specifying printing format in Watchview**

The values of the variables displayed in Watchview can be viewed in octal, hexadecimal, and decimal format.

- **Watch Points are supported through WDB GUI**

The user has to specify variable name and an appropriate value to define a watch point. The user can enter these values from the quickwatch screen of WDB GUI.

- **Support for the `-mapshared` option:**

WDB GUI 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:**

Consistent display of source files in the Source View for FORTRAN applications.

- **Support for WDB Pathmap functionality:**

WDB GUI supports HP WDB debugger 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 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 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 a 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 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 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 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, Step-in, and Step-out support for C and C++:**

- **Support for configuring display of string characters:**

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.

- WDB-GUI enables you to debug your applications in the Command window using the command-line interface.
- **Hexadecimal and Decimal display in Dwell:**

This feature allows you to view the hexadecimal and decimal 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.
- **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 editing:**

Click on the left of Source view or Disassembly view to set or delete a breakpoint. Click on a breakpoint symbol to bring up a dialog box that lets you modify the breakpoint attributes.

- **Program consoles:**

The program consoles supported are `xterm`, `dtterm`, `hpterm`.

- Debugging an executable file or a core file produced by an aborted program
- Attaching to a running process.
- Debugging 32-bit programs.
- Debugging 64-bit programs, supported only on HP-UX 11.x.
- Support for viewing a source file and the corresponding assembly code.
- Searching for specific text and regular expressions in WDB GUI views.
- Support for setting breakpoints and stepping through your program.
- Support for viewing and modifying the values of local and global variables.
- Support for viewing hardware registers.
- Support for traversing the call stack.
- Support for viewing and controlling signal handling.
- Support for traversing threads.
- Support for sending commands directly to WDB using the command line.
- Accessing a history of commands you have executed during the current debug session.
- Support for customizing the appearance and behavior of WDB GUI.
- Support for saving and restoring debugger sessions.

2 Installing WDB GUI

This chapter discusses the installation information for HP WDB GUI.

Installation Overview

The following sections describe the requirement for installing WDB GUI:

Supported Configurations

HP WDB GUI 5.7 supports the debugging of programs written in HP C, HP aC++, and Fortran 90 on Integrity systems running HP-UX 11i v2 or HP-UX 11i v3, and HP 9000 systems running HP-UX 11i v1, HP-UX 11i v2, or HP-UX 11i v3.

Disk Space Requirements

For information on disk space requirements, see the HP WDB release notes that are available on the *HP WDB Download Webpage* at:

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

Installation Instructions

WDB GUI requires HP WDB. If you do not have HP WDB 5.7 installed on your system, it will be loaded automatically when you install WDB GUI 5.7.

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

This chapter describes how to use HP WDB GUI for debugging applications.

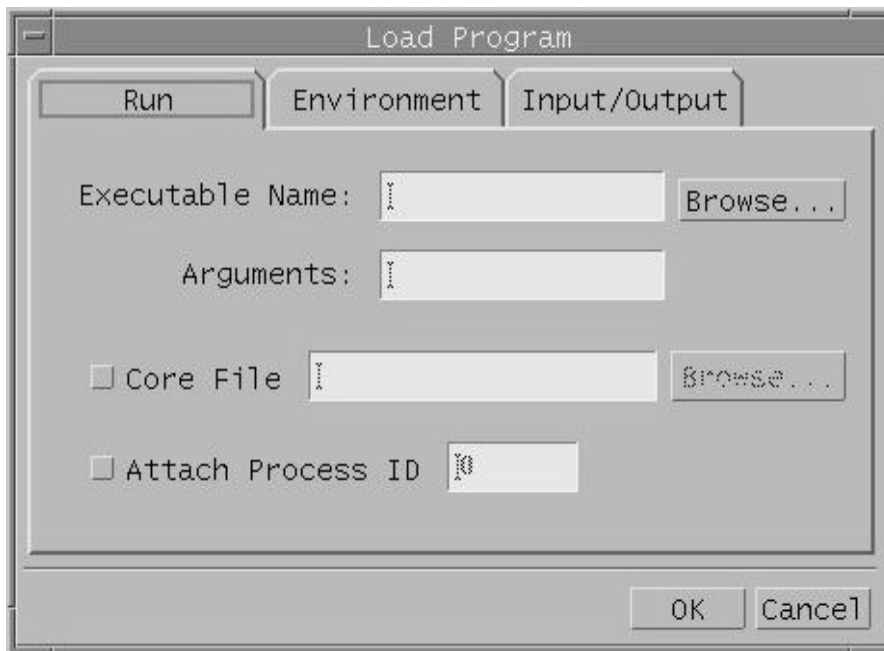
Loading a Program and Changing Program Settings

You can set the following options for the 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 the 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 the 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 the program, run the command `/usr/bin/pxdb -s enable executable_file` where `executable_file` is the name of the program executable.

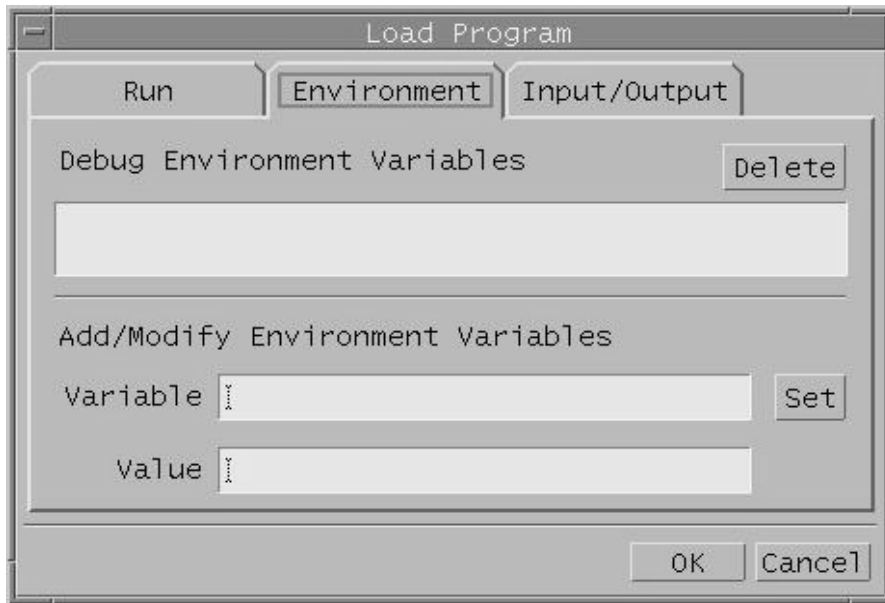
To detach process

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.

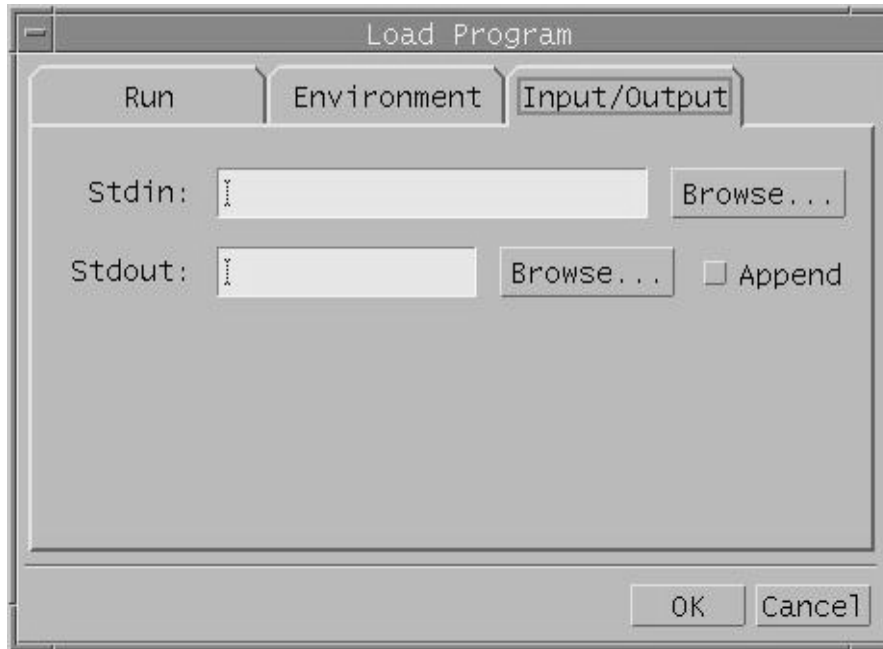
To delete environment variables

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 the program in a terminal window outside the WDB GUI and then attach to the process. This allows you to control standard error output.

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 the 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 the 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)

Field/Button	Action
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.

- 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.

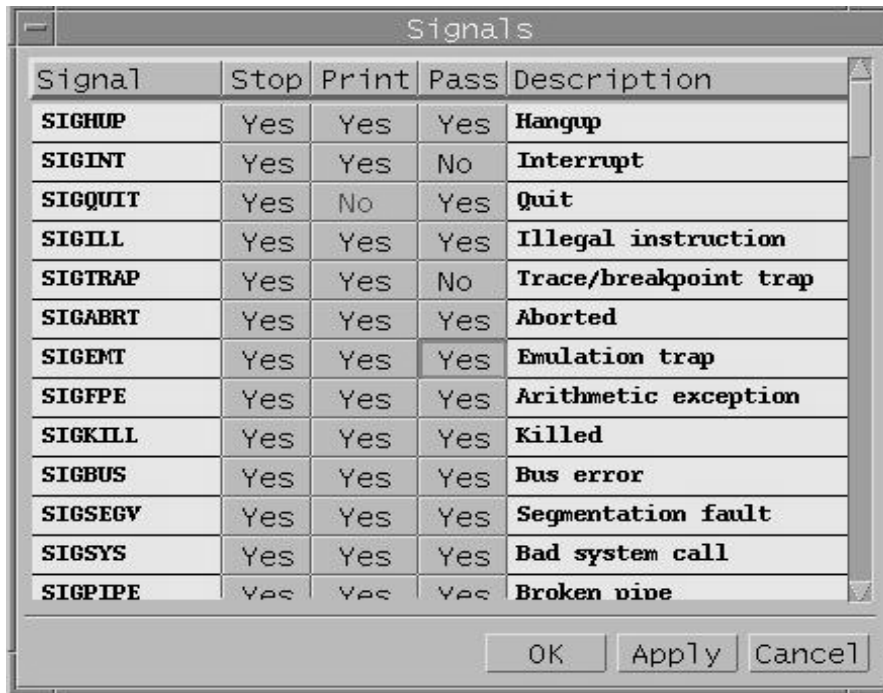


Table 3-2 Columns and descriptions

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
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**

Button	Description
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

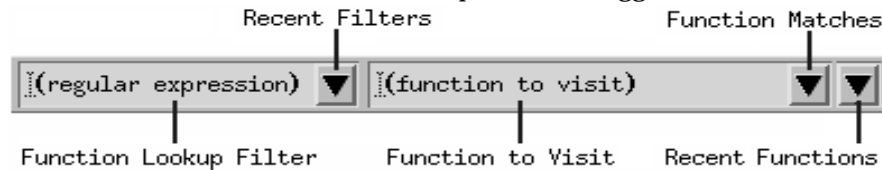
1. On the Edit menu, click Signals.
2. Click on the setting you want to change. Each setting toggles between Yes and No when you click on it.
3. Once you have finished changing the settings, click Apply to commit your changes.
4. Click OK or Cancel to close the dialog box.

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

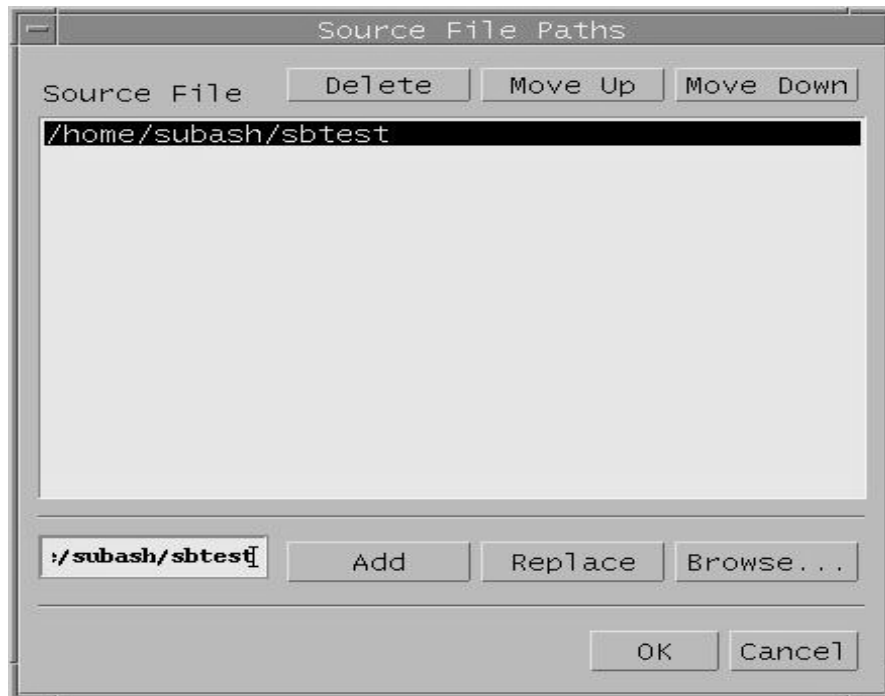
1. On the File menu, click Change Directory.
2. In the Working Directory dialog box, double-click to select the appropriate directory from the Directories list.
3. When the proper directory appears in the Selection box, click OK.

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.

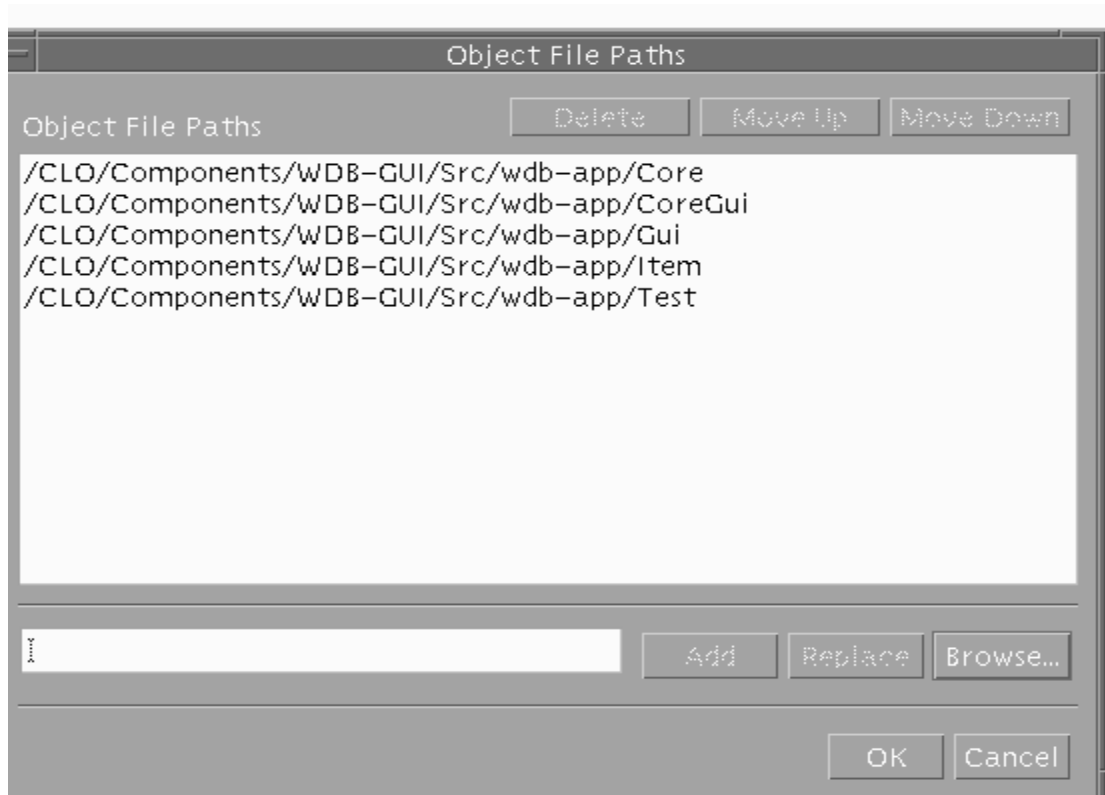
Tips

- 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.

Setting Object Paths

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:

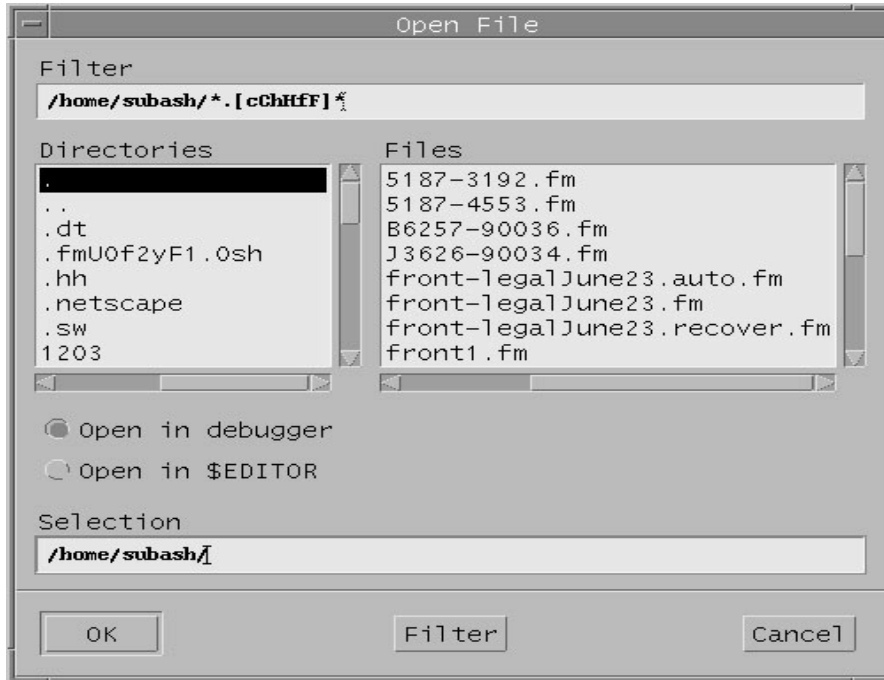


- To change the size of the Recent Files list on the File menu, click Preferences on the Edit menu, and then change the value of Recent files list, maximum size.
- You can open another tab for a file in the Source view by choosing Edit File from the Source view pop-up menu (press M3).
- You can open 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. WDB GUI provides support for the Mouse Scroll Wheel in Source Views and Disassembly Views.



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:

- 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 `dwll`. You can use `dwll` to view the value of individual elements within an array.
- Dwelling on `i` displays the value of `i`.
- Dwelling on `[` displays the value of `a[i]`.

In the expression `*p`, you can use `dwll` 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.

NOTE WDB GUI doesnot support restoring a session and simultaneously attaching it to another process and vice-versa.

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.

Displaying Memory Layout

To display the memory contents for a specific address, complete the following steps

1. Click **View -> Memory Layout**

The Memory Layout tab is dynamically added in the Debug view of the main screen.

2. Enter a valid address in the Address field and click **Enter**

Data is displayed in tabular format in the Memory Layout tab.

NOTE By default, the value of the Display Format is set to hexadecimal with the byte size.

3. Right click to view the options in the Memory Layout

Table 3-3 lists the supported options in the Memory Layout.

Table 3-4 Supported Options

Options	Description
Display Format	This option lists the available display formats.
Size Format	This option lists the available sizes.
Close This View	This option removes the tab from the main screen
Open Separate Window	This option detaches the Memory Layout tab from the main screen and opens it in a separate window

NOTE The formats and the data size can be changed by selecting the **Display Format** and the **Size Format** from the available options.

Viewing Assembly Instruction Description

To view a brief description of the assembly instructions in the Disassembly view, complete the following steps:

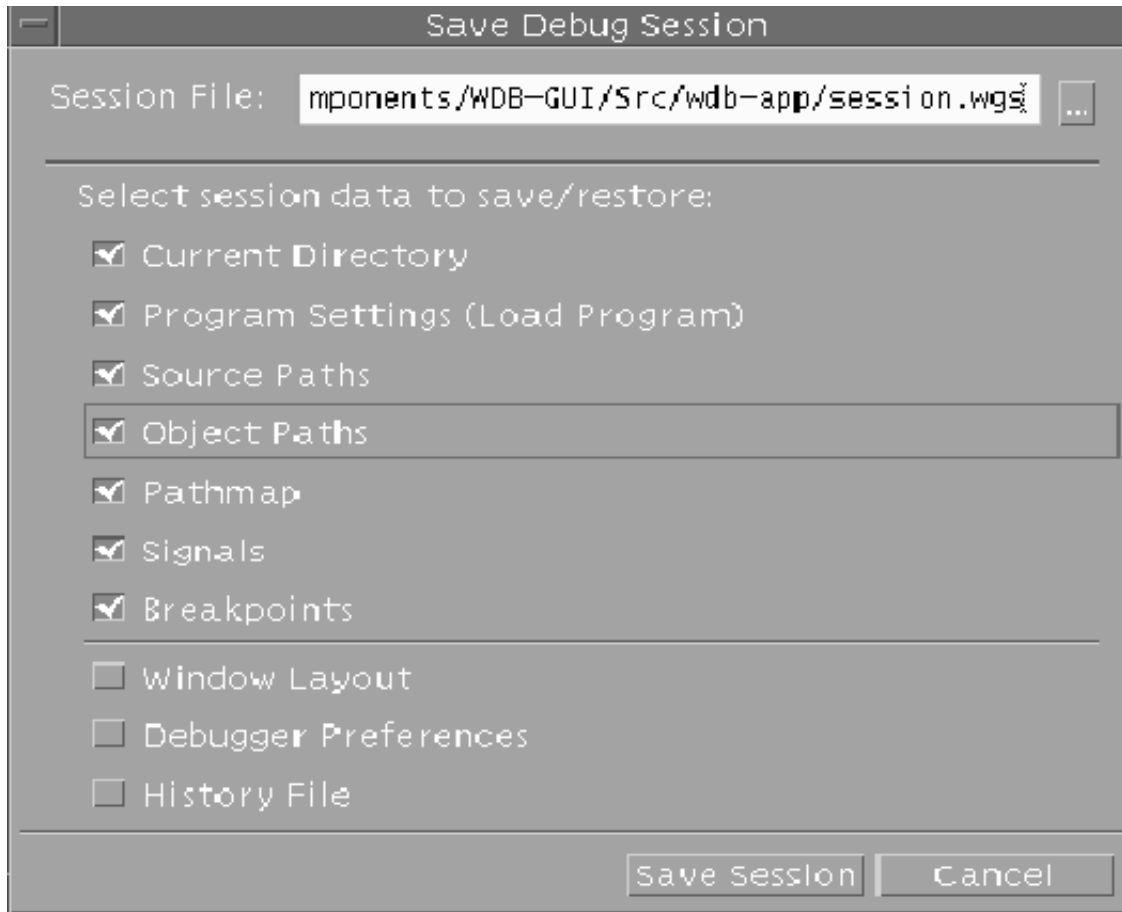
1. Load the program using the **Load Program** option from the main screen
2. Start the debugging session for the program
3. Click the **Disassembly** tab on the main screen.

The assembly dump for the loaded program is displayed.

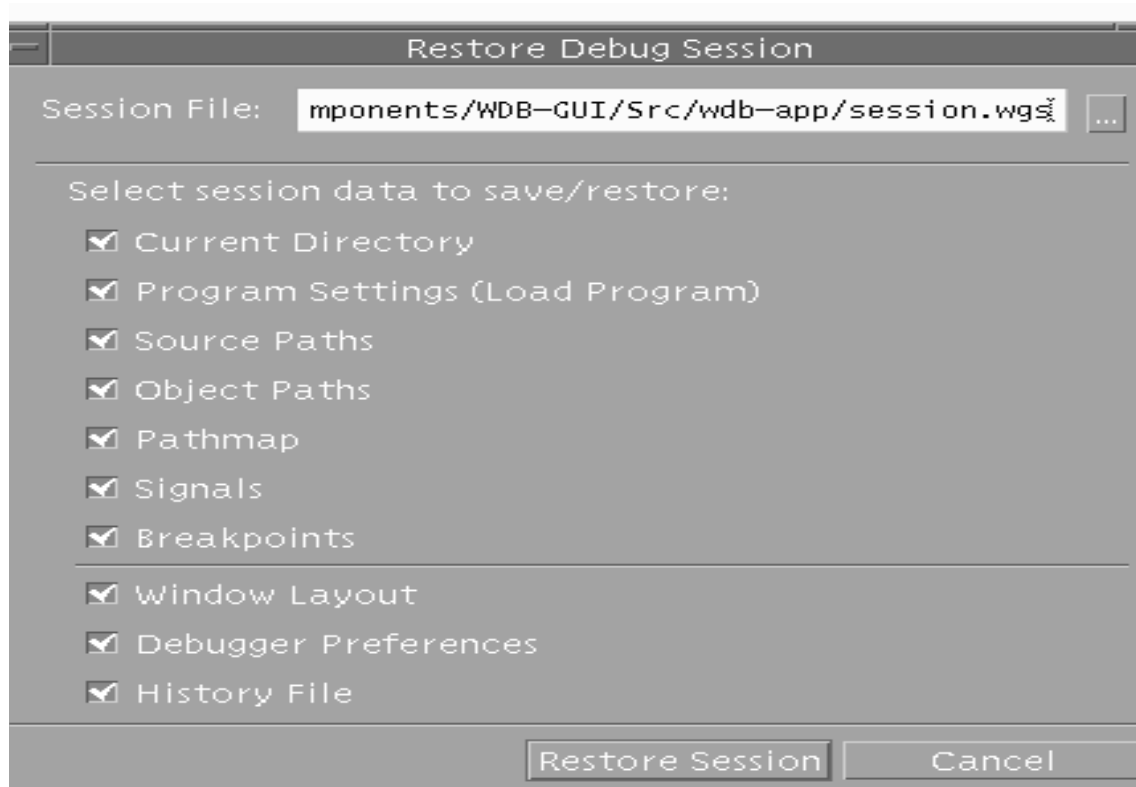
4. Move the mouse pointer over an assembly instruction

A brief description is displayed as a pop-up just below the mouse-pointer.

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.

Debugging Memory Problems

WDB GUI provides support to debug memory-related errors.

Before debugging memory-related errors in a program, you must preload the `librtc.[sl|so]` runtime library for the required program. You can use the `LD_PRELOAD` environment variable or the `+mem_check` option for the `chatr` command to preload `librtc.[sl|so]`.

To enable memory debugging, select **Tools ->Memory Check**. The **Memory Check** dialog window is dynamically displayed. You can enter the memory checking preferences in this window.

NOTE You must enable memory checking and data collection before running your program in the debugger. Additionally, if you change memory checking and data collection settings during a program run, you may need to rerun your program for the new settings to take effect.

Following are the memory checking features, which can be enabled in the **Memory Check** dialog window:

- **RTC heap corruption checks for calls to `strcpy`, `memset`, and `memcpy`**

This feature supports the `set heap-check string [on|off]` command to validate calls to string functions such as `strcpy()`, `strncpy()`, `memcpy()`, `memmove()`, `memset()`, `bzero()`, and `bcopy()`. When this feature is enabled in the **Memory Check** dialog window, the warnings are displayed in the command window.

- **Stop at free of an unallocated or deallocated block address.**

Monitors bad calls to `free()` in the program. Bad calls to `free()` include freeing addresses that have not been allocated or addresses that have been freed. When the debugger detects a bad call to `free()`, it stops the program and reports the bad call in the status bar at the bottom of the debugger window.

- **Stop when freeing a block if bad writes occurred outside block boundary**

When bounds checking is turned on, WDB allocates extra space at the beginning and end of a block during allocation and fills it up with a specific pattern. When blocks are freed, WDB verifies whether these patterns are intact. If they are corrupted, it indicates an underflow or and WDB reports the problem.

- **Scramble previous memory contents on `malloc/free`**

When this setting is turned on, WDB scrambles the space and overwrites it with a specific pattern when a memory block is allocated or deallocated. This change to the memory content, increases the chance that erroneous behaviors will cause the program to fail.

- **Stop if the following block address is allocated or deallocated**

This option can be turned on to stop a program whenever a block at a specified address is allocated or deallocated.

- **Stop when an allocation causes heap growth exceeding num (bytes)**

Monitors the amount of heap growth caused by memory allocations. If the heap grows beyond the specified limit due to a single memory allocation request, the debugger stops the program and reports this in the status bar at the bottom of the debugger window.

- **Stop when an allocation exceeds num (bytes)**

Monitors memory allocations. If the number of bytes allocated by a single allocation request exceeds the specified number of bytes, the debugger stops the program and reports this in the status bar at the bottom of the debugger window.

- **Collect memory leak/usage data (reported in Memory Usage View)**

Causes the debugger to collect data on the program's memory use. When the data is collected, you can view the heap usage data and the memory leak report in the Memory Usage View. The debugger discards the memory data it has collected when the application exits. You can set a breakpoint at program exit to update the memory reports before the application exits.

The memory usage data collected can be filtered with the following options:

- **Report stack trace if block size is greater than number of bytes**

Sets the number of bytes field to one less than the minimum size memory leak or memory allocation that must be reported. Larger values improve run-time performance of the program. Block sizes less than the value specified are still reported, but no stack trace is available.

- **Maximum stack depth to report**

Sets the number in this field to control the number of stack frames that are recorded at each memory allocation. Smaller values improve run-time performance of the program.

- **Incremental Heap Check Settings**

Sets the incremental heap profile settings. Select this option to enable incremental heap profiling. Additionally, you must enter the settings for **Heap Check Interval** and **Heap Check Repeat Count**

Viewing Memory Leaks and Heap Usage

To view the leak report and heap report using HP WDB GUI, complete the following steps:

1. Load the application to the debugger by selecting **File->Load** Program.

In the **Load Program** dialog-box, enter the executable name to load the executable or use the PID to attach the process for debugging.

2. After you load the application, you can set the memory checking preferences by setting the preferences in the **Memory Checking** window. Select **Tools ->Memory Checking** to activate the **Memory Checking** window.
3. Set breakpoints by clicking the rectangular selection strip adjacent to the specific program line-number in the source window. When the breakpoint is set, a red circular button appears at the specified probe-point.
4. To view the heap report and leak reports while debugging the application, select the **Memory Usage** tab in the command window. On selecting the **Memory Usage** tab, the **Memory Leaks** and **Memory Usage** options are displayed.
 - To view the leak report, select the **Memory Leaks** option. The stack-unwind information for each leak can be obtained by expanding the enhanced array browser for each leak.
 - To see a heap report, select the **Memory Usage** option. The stack-unwind information can be obtained by expanding the enhanced array browser for each block of the heap.

Viewing the Runtime Heap and Leak data graphically

To view the graphical representation of the runtime heap and leaks data, Click **Show Graph button** in the **Memory Usage** Tab. The **Memory Usage Viewer (Function Based) window** appears. It depicts the leak and heap data in a Pie Chart. The Pie charts are redrawn based on the selected sorting options which include Bytes, Blocks or Callee.

Viewing the Incremental Heap Profile

HP WDB GUI 5.7 and later versions provide support to view the incremental heap profile for a program.

To view the incremental heap profile for a program, complete the following steps:

- Load the program to WDB GUI.
- Select the **Incremental Heap Check Settings** option while setting the memory debugging preferences in the Memory Check window.
- Select **Tools->Memory Check**.
- Enter **Heap Check Interval** and **Heap Check Repeat Count** in the Memory Check window.
- Run the program after setting the required breakpoints.
- Select **View->Memory Usage->Incremental Heap** to view the incremental heap profile.

The **Incremental Heap View** window displays the incremental heap profile graph for the program. The incremental heap profile graph can be plotted based on the outstanding allocations in the program or the actual heap profile, as follows:

- To view the incremental heap profile graph based on the outstanding allocations, select the **Allocation** option in the **Plot Graph** frame. The **Allocation Profile** displays the outstanding allocations (in KB) in the program with a unique color coding for each interval.
- To view the incremental heap profile graph based on the actual heap profile, select the **Actual Heap** option in the **Plot Graph** frame. The **Heap Space Profile** displays the heap size in KB for the program.

To specify the time interval for displaying the incremental heap profile, you must select the **Select Time** option and specify the start time and the end time from the **Start Time** list menu and the **End Time** list menu. The listings for time in the Start Time list menu and the End Time list menu are calculated by dividing the total program execution time into five equal intervals. Additionally, you can enter a custom start time, or a custom end time for displaying the incremental heap profile.

To view the incremental heap profile summary, click **Summary Table**. The summary table displays the record ID, the start time, the end time, the heap interval, the heap start, the heap end, the heap size in bytes, the number of allocated bytes, and the number of blocks used for all the collected incremental heap profile records. Click on the required incremental heap profile record to view the block allocation details for the corresponding record.

Viewing the Batch RTC Report

WDB-GUI displays Batch RTC Reports on the basis of function names rather than on the basis of block or bytes. You can view Leak, Heap or Corruption reports using the **Batch RTC Report Viewer**.

To view Batch RTC Reports, complete the following steps:

- Select **View->Memory Usage->Batch Report**.
The Batch RTC Report Viewer(Function-based) screen appears.
- Enter a valid Batch RTC generated file (*.leaks/*.heap/*.mem) as Input file. Alternately, use the **Browse** button to locate the file from disk.
- Press **Enter** key.

The Batch RTC report appears in graphical format. There is a Pie Chart depiction of the leak and heap data. The report shows the corruption reports in a tabular format.

The screen provides the following sections of options:

1. **Show Data**

Select the type of information to view as report. The available options are **Leak**, **Heap** and **Corruption**. If the selected input file is of a different type than the chosen **Show Data** type, WDB-GUI considers data from an appropriate file in the same path as the Input file to display report.

2. **Sorted By**

Select the basis on which you want WDB-GUI to sort the data. The available options are Bytes, Blocks or Callee. The Pie charts are redrawn based on sorting order. This option is now applicable for **Corruption** option of **Show Data**.

Viewing the Arena Profile

HP WDB GUI 5.7 and later versions provide support to view the arena information for a program running on HP-UX 11i v3.

To view the arena information, complete the following steps:

- Load the program to HP WDB GUI.
- Stop the program execution at the required breakpoints.
- Select **View->Memory Usage->Heap Arena** to view the arena information. The arena information is displayed in the **View Heap Arena** window.

The following information is displayed in the **View Heap Arena** window:

Arenas

The Arena IDs are listed in the **Arenas** list menu. Select the required Arena ID from the Arenas list menu to view **Arena ID Summary**, or **Block Details** for the selected arena.

Arena ID Summary

The summary information for the selected Arena ID is displayed in the **Arena ID Summary** frame.

Block Details

To view the block level details in an arena, select **Block Details** after selecting the required Arena ID in the **Arenas** list menu. The block distribution in the arena is displayed in the **Arenas Block Distribution** window.

The Arenas Block Distribution window displays the block level space distribution graph for an arena. The graph displays the space occupied by the user blocks, the free blocks, the unclaimed space, and the `malloc()` metadata (which includes the node blocks, the cached blocks, the holding header blocks, and the holding SBA blocks). The virtual address of the blocks is used to arrange the blocks in the graph. The Block ID of the block is also displayed within the block if the scale of the graph supports the display. The start of the heap, the end of the heap, the total heap size, and the total number of blocks are also listed.

HP WDB GUI displays the block distribution graph in the default window. If the complete block distribution graph cannot be displayed in the default window, you must select the **Expand Block Distribution Graph** toggle option to view the magnified block distribution graph.

To view the Block ID, the block type, the block size, and the virtual address for each block in the arena, you must click on the required block in the block distribution graph.

The number of used blocks in each block-count range is also displayed graphically for the selected arena. This information is also displayed in a tabular format.

Heap Arena Space Usage

The **Heap Arena Space Usage** frame displays the Arena ID, the space usage (in KB) in the arena, and the percentage space usage in each arena in comparison to the total space occupied by all the arenas. The comparative space usage across arenas is also displayed in a pie chart. The Arena ID is also displayed in the pie chart if the scale of the graph supports it.

Full Summary

To view the summary for all the arenas, click Full Summary. The summary information for all the arenas is displayed in the Arena Summary window.

Heap Arena Detailed Graphs

The **Heap Arena Detailed Graphs** display the following information:

- The byte distribution (in KB) across the used ordinary blocks, the used small blocks, the free ordinary blocks, and the free small blocks is displayed for each arena. This information is displayed as a bar graph for each arena.

- The byte distribution for the used ordinary blocks, the used small blocks, the free ordinary blocks, and the free small blocks across the arenas is displayed in a pie chart. This information is also displayed in a table.
- The number of blocks that are distributed across the used ordinary blocks, the used small blocks, the free ordinary blocks, and the free small blocks are displayed for each arena. This information is displayed as a bar graph for each arena.
- The number of blocks occupied by the used ordinary blocks, the used small blocks, the free ordinary blocks, and the free small blocks across the arenas are displayed in a pie chart. This information is also displayed in a table.

Enabling and Disabling Specific Threads

You can explicitly enable or disable the execution of specific threads when debugging multi-threaded applications. The following steps discuss how to enable or disable the execution of specific threads:

1. Select the **Threads** tab to enable the **Threads View**. The **Threads View** displays the current threads that are active in the program that is being debugged.
2. To prevent the execution of a specific thread until it is enabled again, right-click on the specific thread and select the **Disable Thread** option. A disabled thread is indicated by a 'D' symbol before the thread-id.
3. To enable the execution of a specific thread when you step or continue, right click on the specific thread and select the **Enable Thread** option. By default, all threads are enabled.

Advanced Thread Debugging Support

WDB GUI 5.6 and later versions provide advanced thread debugging support to view the state of `pthread` primitives in applications running on 11i v2, or 11i v3 operating systems. You can view extended information on the state of threads, read-write locks, mutexes and conditional variables.

Prerequisites for Advanced Thread Debugging in WDB GUI

The following pre-requisites apply for advanced thread debugging in WDB GUI:

- The advanced thread debugging features are supported for PA-RISC and Itanium applications running on HP-UX 11i v2 and later versions of the operating system. HP WDB GUI does not support thread debugging of PA-RISC applications on Integrity systems
- The tracing `pthread` library is required for advanced thread-debugging. The `pthread` tracer library is available by default on systems running HP-UX 11i v2 or later.
- The thread debugging feature in HP WDB is dependent on the availability of the Dynamic Linker Version B.11.19.
- HP WDB GUI uses `librtc.sl` to enable thread debugging support. If the debugger is installed in a directory other than the default `/opt/langtools/bin` directory, you must use the environment variable, `LIBRTC_SERVER`, to export the path of the appropriate version of `librtc.sl`.
- HP WDB GUI does not support debugging of programs that link with the archive version of the standard C library, `libc.a`, or the core library, `libcl.a`. The programs must be linked with `libc.sl`.
- For PA-RISC 32-bit applications, the dynamic library path look-up must be enabled for advanced thread debugging. To enable dynamic library path look-up for advanced thread debugging, enter the following command at HP-UX prompt:

```
# chatr +s enable <PA32-bitApp>
```

This command automatically enables dynamic library path look-up. No additional environmental variables are required.

Support for `info thread`, `info thread <thread-id>` and `set thread check <on/off>`

To view information on the state of the current threads in the application, complete the following steps:

1. To view information about the threads, select the **Threads** tab. This displays all the current threads in the application that is being debugged.
2. To enable advanced thread debugging, right click on the Threads view and select the **Advanced Thread View** option. This dynamically adds the **Mutex** tab, the **R/W Locks** tab, and the **Cond Var** tab to the debug view. You can also select **Tools->Advanced Thread View** to enable advanced thread debugging.

The **Advanced Thread View** must be enabled before loading the application for viewing information on individual threads and pthread primitives like read-write locks, mutexes and conditional variables.

3. To view extended information on a specific thread, right-click on the required thread and select the **Thread Info** option.

Support for info mutex and info mutex <mutex-id>

The **Advanced Thread View** must be enabled before loading the application for viewing extended information on mutexes.

To view information on the state of the current mutexes in the application, complete the following steps:

1. To view information about mutexes, select the **Mutex** tab. This displays all the current mutexes in the application that is being debugged. To view this information in a new window, right-click on the **Mutex View** and select the **Open Separate Window** option.
2. To view information about a specific mutex, right click on the required mutex and select the **Mutex/Thread Info** option. This information is displayed in a separate window.

Support for info rwlock and info rwlock <rwlock-id>

The **Advanced Thread View** must be enabled before loading the application for viewing extended information on in read write locks.

To view information on the state of the current read-write locks in the application, complete the following steps:

1. To view information about read-write locks, select the **R/W Locks** tab. This displays all the current read-write locks in the application that is being debugged. To view this information in a new window, right-click on the **R/W Locks View** and select the **Open Separate Window** option.
2. To view information about a specific read-write lock, right click on the required read-write lock and select the **RW Lock /thread Info** option. This information is displayed in a new window.

Support for info condvar and info condvar <condvar-id>

The **Advanced Thread View** must be enabled before loading the application for viewing extended information on conditional variables.

To view information on the state of the current conditional variables in the application, complete the following steps:

1. To view information about conditional variables, select the **Cond Var** tab. This displays all the current current conditional variables in the application that is being debugged. To view this information in a new window, right-click on the **Cond Var View** and select the **Open Separate Window** option.
2. To view information about a specific conditional variable, right click on the required conditional variable and select the **Condvar/Mutex Info** option. This information is displayed in a new window.

Debugging Inline Functions

HP WDB GUI 5.6 and later versions provide inline debugging support for debugging applications running on Integrity and HP 9000 systems. To enable inline debugging in HP 9000 systems, the applications must be compiled with the `+inline_debug` option (added in the A.03.65 and later versions of the HP aC++ compiler). In Integrity systems, the applications that are compiled with the `-g` option support inline debugging by default. Compiler versions A.06.02 and later in Integrity systems support inline debugging.

Using Inline Debugging Options

To enable or disable inline debugging, select **Tools->Inline Debug** and set the appropriate inline debugging option. **Inline Debug** offers the following options:

- **On**
Select **Tools->Inline Debug->On** to enable inline debugging functionality without the inline breakpoint feature. This command is enabled by default in Integrity and HP 9000 systems.
- **Off**
Select **Tools->Inline Debug->Off** to disable the inline debugging feature. You can disable inline debugging by entering this command before loading the application for debugging.
- **Breakpoint All**
Select **Tools->Inline Debug->Breakpoint All** to enable the setting and modifying of breakpoints at all instances of a particular inline function, in addition to enabling the inline debugging feature. A single instance of the specified inline function is displayed as a representative instance for all the instances of the specified inline function. This creates a single-breakpoint illusion for multiple instances of the inline function. You can set and modify breakpoints at all instances of the inline functions by setting and modifying breakpoints at the displayed instance of the inline function. This option must be enabled before loading the application for debugging. This option is not available for HP 9000 systems.
- **Breakpoint Ind.**
Select **Tools->Inline Debug->Breakpoint Ind.** to enable the setting and modifying normal breakpoints at a particular inline function, in addition to enabling the inline debugging feature. All instances of a particular inline function are displayed as separate

breakpoint occurrences. You can set or modify individual breakpoints at specific instances of the inline function, while retaining the breakpoints in other instances of the specified inline function. This option must be enabled before loading the application for debugging. This option is not available for HP 9000 systems.

Setting Breakpoints on Inline Functions

To set breakpoints on Inline Functions, complete the following steps:

1. Select **Edit->Breakpoints** to display the **Breakpoints** window
2. Enter the breakpoint information in the **Breakpoints** window.

Limitations for Inline Debugging

- The inline breakpoint features are not available for HP 9000 systems.
- In Integrity systems, the inline breakpoint feature is not available for programs that are compiled with +O2 optimization level and above.
- The inline breakpoint features can degrade performance of the application that is being debugged. You can explicitly disable the breakpoint features and continue to use other inline debugging features if these features are not required.
- Inline breakpoint features are not supported for PA-RISC applications running on Integrity systems.

Viewing the Execution Path Entries

HP WDB GUI 5.7 and later versions of the debugger enable you to view the execution path entries for programs running on Integrity systems. This feature enables the display of the execution path taken across branched modules.

Compiler Dependencies

This feature is supported only for compiler versions A.06.15 and later.

You must compile the program with `+pathtrace` compiler option to view the execution path entries.

The `+pathtrace` compiler option provides a mechanism to record program execution control flow into global path tables, local path tables, or both. This saved information enables the debugger to display the execution path entries for the program. To print the execution path entries in the current thread or frame for programs running on Integrity systems, you can set the required sub-options for the `+pathtrace` compiler option.

The following sub-options are available for the `+pathtrace` compiler option:

```
+pathtrace= [<global|global_fixed_size>:<local>]
```

For more information on the `+pathtrace` compiler option, see the *aCC(1)* manpages.

Displaying the Execution Path Entries

To view the execution path entries, complete the following steps:

1. Compile the program with the `+pathtrace` option, as follows:

```
cc -g +pathtrace <executable>
```

2. Load the program to HP WDB GUI
3. Set the required breakpoints and run the program
4. When program execution stops, select **View->Execution Path**

The execution path entries are displayed in the **View Branch Execution Path** window. The execution path entries are displayed upto the current instant of program execution.

The **View Branch Execution Path** window displays the following information:

- **Local/Current Execution Path Table (LEPT)**

This table displays all the branch path entries within the current function.

The LEPT frame displays the following information:

- The serial number for each local execution path entry
- The corresponding PC address for the first instruction in each branch that is executed within the current function
- The branch ID is displayed exclusively for each branch module that is executed. However, all the branch modules within the function are assigned an branch ID. (For example, the if statement and the else statement are each assigned unique IDs. However only the ID of the executed branch, which is the ID of the if statement or the ID of the else statement is displayed).

The LEPT frame also displays the total branch paths found, the total number of branches executed, the Branch ID of the last branch executed, the current function, and the current filename.

By default all local execution path entries are highlighted in the source view and the disassembly view. To exclusively highlight a specific local execution path table entry in the source view and the disassembly view, click on the required local execution path entry. To revert back to highlighting all the local execution path entries in the source view and disassembly view, click on **Highlight all LEPTs**.

- **Global Execution Path Table (GEPT)**

The GEPT frame displays the global execution path entries for each of the branches in the program execution path.

The GEPT table displays the following information:

- The number of branch paths found
- The serial number for each global execution path entry
- The name of the file, where the first instruction in the executed branch occurs
- The line number of the first instruction in each branch module of the program execution path
- The function name, where the first instruction in the executed branch occurs
- The PC address of the first instruction in the corresponding executed branch

To highlight the global execution path entry in the source view and the disassembly view, click on the required global execution path entry. To revert back to the default state where all local execution path entries are highlighted in the source view and the disassembly view, click on **Highlight all LEPTs**.

If the number of global execution path entries exceed 200, all the entries are not displayed in the same GEPT frame. You can click on First, Next, Previous, and Last to traverse and view all the global execution path entries

NOTE WDB GUI truncates the filename and function name to a maximum of 20 characters. If the filename or the function name exceeds 20 characters, you can click on the corresponding global execution path entry to view the complete file name, the name of the function, the line number and the PC address of the first instruction in the corresponding branch.

4 Troubleshooting WDB GUI

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

Table 4-1 Error Message descriptions (Continued)

Message	Cause	Solution
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"> • The backend is in the process of connecting to the WDB GUI. • Or the backend has unexpectedly gone away 	<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)

Message	Cause	Solution
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.

Known Problems

- **Problem:**gdb prompt gets disabled in WDB-GUI

Workaround:Giving a `on` on gdb prompt in WDB-GUI, and disables gdb prompt.

- **Problem:**WDB GUI Opens Multiple Windows

Workaround:WDB GUI opens the same source file twice in the source window, when TERM the working directory is changed to an NFS mounted directory which has a canonical reference. For example, `/tmp` can canonically refer to `/disk3/tmp`.

You might encounter the problem if you follow these steps:

- Change the working directory of WDB GUI to some NFS mounted directory which has canonical reference (`cd /tmp`). WDB GUI prints this message in the Command window:

```
(gdb) cd /tmp
Working directory /tmp
(canonically /disk3/tmp)
```

- Load an executable in the GUI and step into the code:

```
(gdb) file a.out
(gdb) b main
Breakpoint 1 at 0x2f24: file /tmp/a.c, line 44
(gdb) r
Starting program: /tmp/a.out
(gdb) s
```

- Now open the source file of the executable, say `a.c`, in both of the following two ways:

1. Right-click in the Source window and select Edit File from the Source popup menu.
2. Use the Open File option in the Tool bar menu

You will find that the same file will be opened twice in the Source window from the following locations:

- `/disk3/tmp/a.c`
- `/tmp/a.c`

Known Problems

If you modify and save the source file in one window, the changes will not be visible in the second window. If you then modify the source file in the second window, WDB GUI will display this warning:

The file has been modified since opened or last saved. Save anyway? This happens only for an NFS mounted filesystem that has a canonical reference and you do a `cd` to that directory from within the debugger.

- **Problem:Terminal Window Sometimes Fails to Start on 10.20**

Workaround:The WDB GUI uses a terminal window to display the target programs output and to allow user input to the target. The terminal program is started when the **View:Program Console** menu pick is chosen or when the target program generates any output. Some versions of `dtterm` on 10.20 cannot be started from the WDB GUI. If this happens you will see the error:

```
Unable to start terminal
```

This is caused by a problem in `dtterm`, which is fixed in patch `PHSS_17566`.

If you donot have the patch available, you can work around the problem by setting the environment variable `TERM` to `hpterm` and restarting the WDB GUI. You can also work around the problem by starting the target program from a shell window and then attaching to the program from the WDB GUI.

- **Problem:Attaching to a Process May Generate Invalid Error**

Workaround: If you attach to a program with a command line argument when you start the WDB GUI, you may see an error that says:

```
No such file or directory.
```

This error may be displayed even when the WDB GUI was successful in attaching to the process. If the WDB GUI comes up with the process loaded, you may ignore this message.

NOTE This problem does not occur if you start the WDB GUI and then attach to a process from the WDB GUI menu.

- **Problem:Viewing Large Call Stacks**

Workaround:Runaway recursive functions can cause the call stack to become very large. To prevent the long delays that would result when displaying a very large call stack, the number of stack-frames displayed is limited to 200. To view a call stack with more than 200 frames, use the `backtrace` command from the **Command** view.

- **Problem:Problems in invoking Program Console with `.cshrc`**

Workaround:

1. If you have a read command in the `.cshrc` file and if you have not specified the absolute path for the Program Console (ie) `TERM=dtterm`, and if you try to open the Program Console, the GUI will hang or will dump core.

When there is no full path specified for the `dtterm/ hpterm/xterm`, WDB GUI will use the `which` command to find the path for these terminals. `which` executes the `.cshrc` file and the read in `.cshrc` waits for some input . But since WDB GUI has no associated terminal for input, it will hang or dump core.

To solve this, remove/comment the 'read' in the `.cshrc` file while invoking WDB GUI.

2. You might find the following warning appearing on the Status Line, when you invoke the GUI's program console :

```
Note: Trying other terms ( could not find $TERM dtterm )
```

The warning will appear if the `PATH` for the term is not set properly. To avoid getting this warning, set the absolute path for `hpterm /dtterm /xterm` in `TERM`. For example:

```
$ export TERM=/usr/bin/X11/dtterm
```

3. WDB GUI will coredump if `TERM` variable is not initialised. For example:

```
$ export TERM=
```

4. To avoid the coredump, set `TERM` appropriately.

- Problem:Fonts problem in non-HP-UX machines

Workaround:WDB GUI might dump core when you set the `DISPLAY` environment variable to a non-HP-UX machine say `SUN` or `LINUX`. The non-HP-UX machines do not recognise HP fonts and so the application will dump core.

The solution is to copy the HP fonts on the non-HP-UX machine and to set that directory in the `fontpath`. All necessary HP fonts reside in the following directory in the HP-UX machines:

Create a tar file, say `fonts.tar` of this directory and copy the tar file to the non-HP-UX machine.

At the non-HP-UX machine, before invoking the WDB GUI,

1. Goto the directory where you have copied the tar file, say `/tmp`
2. Untar the tar file
3. `xset +fp /tmp`
4. `xset fp rehash`

Another possible cause of coredump might be the absence of the motif patches PHSS_22947 and PHSS_23823. You can install these patches as well, to solve the coredump problem.

- **Problem:**Support of FORTRAN array slices 0

Workaround:HP WDB-GUI 3.1.5 or higher does not support FORTRAN array slice feature in the Local Variables, Watch, and Quick Watch windows. The HP WDB 3.1.5 or higher command-line supports this feature. However, the user can make use of the GDB command line prompt to print the FORTRAN array slices.

The WDB GUI 5.5.2 does NOT support the following features on Integrity systems:

- **Fix and continue:** Edit sources and have the debugger compile and patch in the changes without restarting the debugged program.
- **Steplast support for C and C++ in PA:** In general, if a function call has arguments which makes further function calls, a simple step command will step into argument evaluation call. This new feature, `steplast`, will step into a function but not into the calls for evaluating arguments.

NOTE The `steplast` is supported in WDB GUI with HP WDB version 3.1.5 or higher.

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 web browser set in the Edit->Preferences. If the default command fails, the WDB GUI will issue a warning and allow you to edit the browser command or allows you to select the different browser.

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>