ULTRIX
Worksystem Software

digital

Guide to the dxdb Debugger

Order Number: AA–MA93B–TE

ULTRIX Worksystem Software

Guide to the dxdb Debugger

Order Number: AA-MA93B-TE

Product Version:	ULTRIX Worksystem Software, Version 2.2
Operating System and Version:	ULTRIX–32, Version 3.1 or higher

digital equipment corporation
maynard, massachusetts

CDA	DTIF	VAXstation
DEC	MASSBUS	VMS
DECUS	MicroVAX	VMS/ULTRIX Connection
DECnet	Q-bus	VT
DECstation	ULTRIX	XUI
DECwindows	ULTRIX Mail Connection	digital
DDIF	ULTRIX Worksystem Software	
DDIS	VAX	

This manual was written and produced by the Open Software Publications group.

Contents

Figures

Tables

About This Manual

The *Guide to the dxdb Debugger* describes how to use the various features of the dxdb debugger.

Audience

This guide is written for programmers with both a working knowledge of general debugging techniques and a basic knowledge of the DECwindows interface.

Organization

The *Guide to the dxdb Debugger* contains the following sections:

Introduction
: Illustrates the various dxdb windows and commands and describes the routines diroll.c and didisp.c.

Starting dxdb
: Describes how to invoke the debugger.

Viewing a Source File
: Describes how to open and display a file in the dxdb Source window.

Running a Program
: Describes how to run a program in dxdb and shows the Input/Output Window.

Setting Breakpoints and Conditional Breakpoints
: Describes how to set and delete a breakpoint and a conditional breakpoint in the Breakpoint window.

Setting Tracepoints and Conditional Breakpoints
: Describes how to set and delete a tracepoint and a conditional tracepoint in the Breakpoint window.

Methods for Selecting Text
: Describes how to print, examine, and delete the value of program variables.

Controlling Program Execution
: Shows how to use the Step and Skip windows to execute a program in single- or multiple-line increments.

Assigning a Value to a Variable
: Explains how to assign a value to a specific program variable.

Exiting From dxdb
: Describes how to exit from dxdb.

Supplying Arguments to a Program
: Describes how to use the Run window to supply an argument to a program.

Viewing the Program Execution Stack
: Shows how to use the Stack window to view elements on the program execution stack.

Viewing Variables
: Shows how to use the Dump window to view the values of all currently active local variables.

Getting Information About Variables
: Describes how to use the Whatis, Which, and Whereis functions to obtain the data type, scope, and location of a particular variable.

Editing Files
: Describes how to edit the file you are currently displaying.

Restarting the Debugger
: Describes how to rebuild an executable program and restart the debugging session.

Related Documents

Introduction to the ULTRIX Worksystem Software User Environment
: Provides an overview to ULTRIX Worksystem Software utilities.

DECwindows User's Guide
: Provides a discussion on basic DECwindows functions and provides information on customizing DECwindows applications.

Conventions

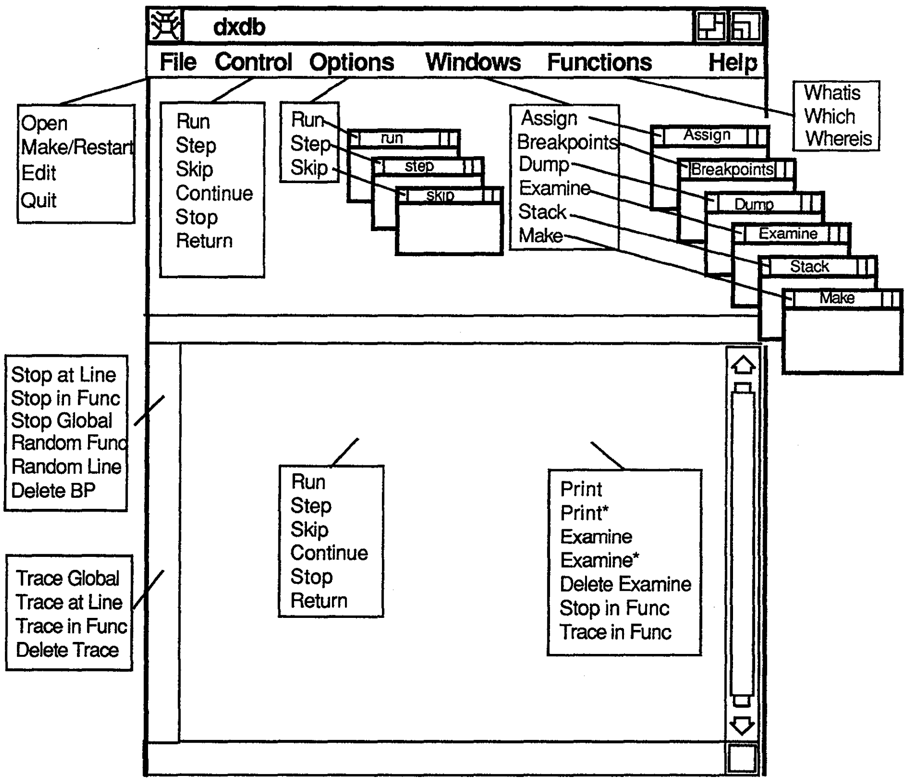
The following conventions are used in this manual:

%	The default user prompt is your system name followed by a right angle bracket. In this manual, a percent sign (%) is used to represent this prompt.
`user input`	This bold typeface is used in interactive examples to indicate typed user input.
`system output`	This typeface is used in interactive examples to indicate system output, and also in code examples and other screen displays. In text, this typeface is used to indicate the exact name of a command, option, partition, pathname, directory, or file.
UPPERCASE lowercase	The system differentiates between lowercase and uppercase characters. Literal strings that appear in text, example, syntax descriptions, and function definitions must be typed exactly as shown.
rlogin	In syntax descriptions and function definitions, this typeface is used to indicate terms that you must type exactly as shown.
filename	In syntax descriptions and function definitions, italics are used to indicate variable values; and in text, to introduce new terms or give references to other documents.
`cat(1)`	Cross-references to the *ULTRIX Reference Pages* include the appropriate section number in parentheses. For example, a reference to `cat(1)` indicates that you can find the material on the `cat` command in Section 1 of the reference pages.
<KEYNAME>	This symbol is used in examples to indicate that you must press the named key on the keyboard.
MB1,MB2,MB3	Unless the mouse buttons have been redefined, MB1 indicates the left mouse button, MB2 indicates the middle mouse button, and MB3 indicates the right mouse button.

Guide to the dxdb Debugger

The dxdb debugger is a DECwindows utility that provides you with a versatile environment in which to debug programs. The windows and menus of dxdb contain all the commands that you will need during a typical debugging session. Figure 1 shows the windows and commands available in dxdb.

Figure 1: The dxdb Debugger Windows and Commands



ZK-0039U-R

1.1 Introduction

This guide introduces basic dxdb operations by showing a dxdb debugging session on a sample C program, diroll, and by focusing on specific dxdb operations.

Diroll is a simple dice-rolling program that consists of two routines: diroll.c and didisp.c. The main routine, diroll.c, determines the values of dice variables die1 and die2. Diroll obtains these values from the C random number generator (random) and then passes them to the external routine didisp.c. The uncorrected version of diroll consists of the following code:

```
/***  diroll.c  ***/

char roll = ' ';
int diel, die2;

main()    /*Dice roll example program*/
{
    char s[512];

    srandom (getpid());      /*Set random seed using the return
                               from standard c function getpid().*/

    printf("Press <CR> to roll...");
    gets(s);

    for(;;) {

        diel = random() %7;     /*Get random numbers*/
        die2 = random() %7;

        didisp(diel,die2);      /*Draw dice*/

        printf("Press <CR> to roll...");  /*Begin the game*/
        gets(s);
    }
}
```

Didisp.c uses two matrices to define the dots and the faces of each die. Didisp uses the values of die1 and die2 as offsets into these matrices to display each die. The uncorrected version of didisp.c consists of the following code:

```
/***  didisp.c  ***/

char *dots[6] = {                        /*Define dot combinations*/
     "|  o  |",
     "|o    |",
     "|    o|",
     "|o   o|",
     "|     |",
     "+ - - +"
};

int faces[6][5] = {                      /*Define dice faces*/
     {5, 4, 0, 4, 5},
     {5, 1, 4, 2, 5},
     {5, 1, 0, 2, 5},
     {5, 3, 4, 3, 5},
     {5, 3, 0, 3, 5},
     {5, 3, 3, 3, 5}
};

int die1, die2;

didisp (die1, die2)                     /*Function that draws dice.*/
{
    int i, j, k;

    for (i = 0; i < 5; i++) {  /*Index into arrays and draw dice*/
        j = faces[die1][i];
        k = faces[die2][i];
        printf("%s     %s0, dots[j], dots[k]);
    }
return;
}
```

When you run diroll, the program should generate output similar to the following:

```
Press <CR> to roll...
+ - - +   + - - +
|o    |   |     |
|  o  |   |  o  |
|    o|   |     |
+ - - +   + - - +
Press <CR> to roll again.
+ - - +   + - - +
|o    |   |o   o|
|  o  |   |  o  |
|    o|   |o   o|
+ - - +   + - - +
Press <CR> to roll again.
```

1.2 Starting the dxdb Debugger

You invoke the dxdb debugger using the following syntax:

dxdb *file*

The file argument must be an executable C file that was compiled with the -g option of the C compiler command (cc). For example, to run dxdb on diroll you must have first compiled diroll.c and didisp.c using a cc command similar to the one that follows:

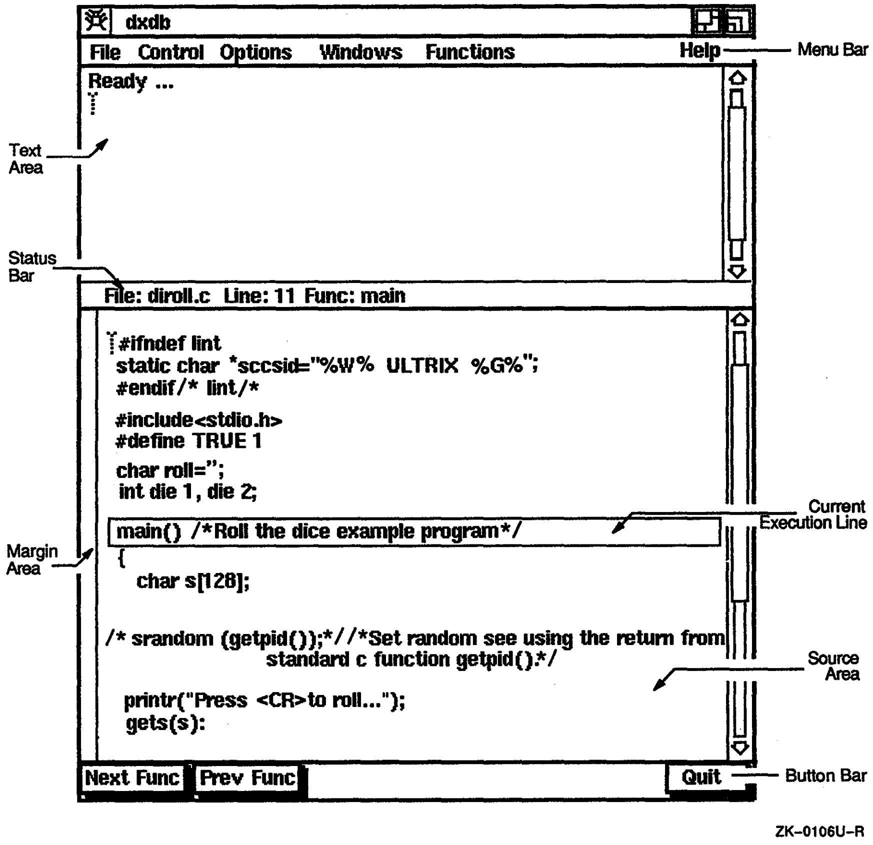
```
% cc -g diroll.c didisp.c -o diroll
```

You invoke the dxdb debugger as follows:

```
% dxdb diroll
```

When invoked, dxdb displays the Control window. Figure 2 shows the Control window.

Figure 2: The Control Window



```
Ready ...
```

```
File: diroll.c  Line: 11  Func: main
```

```
#ifndef lint
static char *sccsid="%W% ULTRIX %G%";
#endif/* lint/*
#include<stdio.h>
#define TRUE 1
char roll='';
int die 1, die 2;
main() /*Roll the dice example program*/
{
  char s[128];

/* srandom (getpid());*//*Set random see using the return from
                    standard c function getpid().*/
  printr("Press <CR>to roll...");
  gets(s):
```

By default, dxdb displays only the Control window. However, you can change this default (and display several windows on startup) by modifying the .Xdefaults file (see X(1X) in the *ULTRIX Worksystem Software Reference Pages*).

You can use the Control window to access all other dxdb windows and commands. Table 1 describes the Control window areas.

Table 1: Control Window Area Summary

Window Area	Description
Menu Bar	Contains menus of items that let you run and debug programs. To choose an item, point to the menu, press and hold MB1, and drag the pointer through the menu. Release MB1 on the desired item.
Text Area	Read-only area that dxdb uses to display run-time and error messages. You can scroll through the contents of this area using the scroll bar.
Source Area	Read-only area that displays the source code being executed. You can use this area to display other source files of interest.
Status Bar	Displays the current file, line, and function of execution.
Margin Area	Shows breakpoints and tracepoints. Provides menus for setting and deleting breakpoints and tracepoints.
Button Bar	Contains command buttons that you can use to manipulate the Source Area.
Popup Menus	Contains the Conrol, Examie, Breakpoints, and Tracepoints menus. You can invoke these menus by pressing MB2 or (shift-MB2) in the Source Area or the Margin Area.
Current Execution Line	Contains the current line of execution and is signified by placing a box around the text.

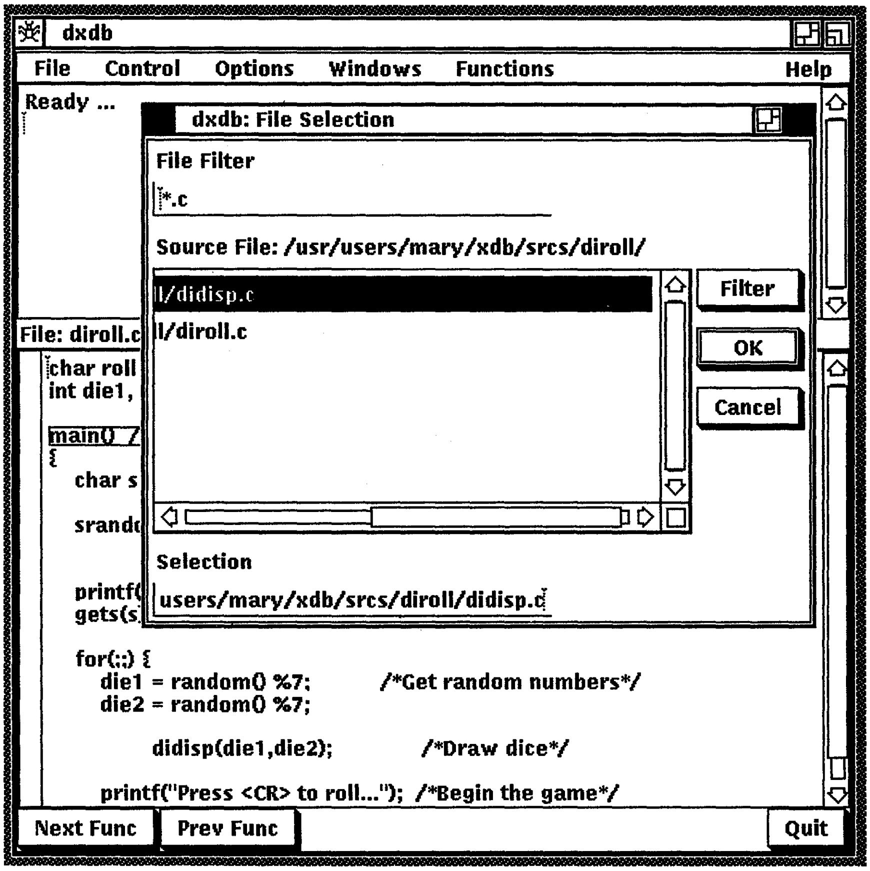
1.3 Viewing a Source File

At the top of the Source Area, is the Status Bar, which indicates the file that is currently displayed in the window, the current line the debugger is on, and the name of the function in which that line resides.

1.3.1 Opening and Closing Files

To open and display a file in the Source Area, choose Open from the File menu of the Control window. The debugger then displays the File Selection window. Figure 3 shows the File Selection window.

Figure 3: The Source Area with Source File Window



```
char roll
int die1,

main() /
{
   char s

   srand(

   printf(
   gets(s

   for(;;) {
      die1 = random() %7;       /*Get random numbers*/
      die2 = random() %7;

            didisp(die1,die2);        /*Draw dice*/

      printf("Press <CR> to roll..."); /*Begin the game*/
```

The text area of the File Selection window contains a listing of all C files in your current directory. You can use the scroll bar to scroll the listing up or down.

Directly above the text area is a message that indicates the current working directory. To open a file in this directory, click on the Selection area and type the file name, followed by a carriage return. The debugger displays the file in the Source Area. To remove the File Selection window, click on the Cancel button. Figure 3 shows the selection of the file didisp.c.

You can also open a file by clicking on the file in the text area (this file is automatically inserted at the Selection prompt) and then clicking on the OK button.

To open a file located in another directory, enter the appropriate directory and file at the Selection area. For example, if you enter xdir/t.c, the file t.c located in the xdir directory is opened.

If you want to display the entire contents of directory xdir in the text area of the window, click on the File filter prompt (at the top of window), enter the directory, and click on the Filter button. You can then choose a file using one of the methods previously described. To cancel a current file selection, click on the Cancel button.

1.3.2 Browsing Through Source Files

The Button Bar allows you to perform two browsing operations (Next Func and Prev Func) on any file displayed in the text area of the Source Area. Next Func causes dxdb to advance to the next function defined in the source code. Prev Func causes dxdb to return to a previous function.

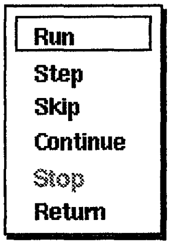
1.4 Running a Program

Because the program diroll is known to contain bugs, you will need to run the program under dxdb control to uncover where the code is failing. The debugger allows you to run a program conditionally (passing parameters to the routine) or unconditionally (no parameters passed to the routine).

Because there are no parameters to pass to diroll, you must run the program unconditionally. You can do this in two ways:

1. Choose the Run option from the Control option in the Menu Bar.
2. Click MB2 in the Source Area and choose the Run option from the Control menu. Figure 4 shows the Control menu.

Figure 4: The Control Menu

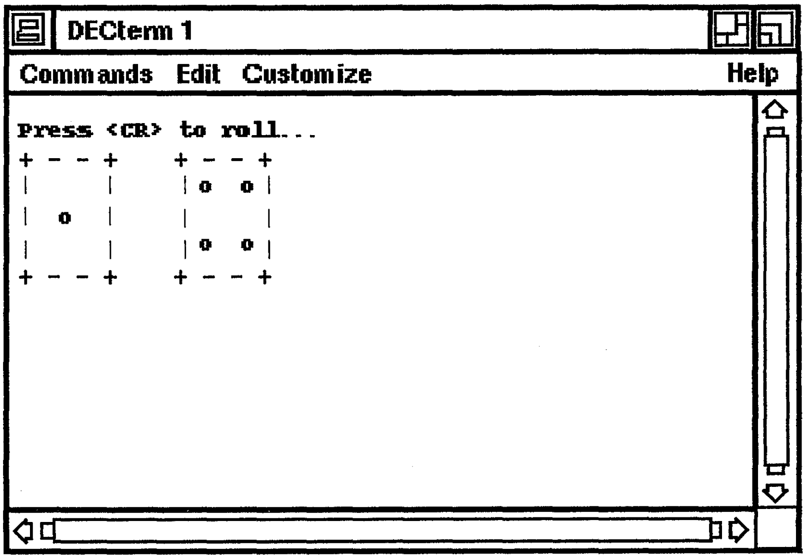


ZK-0107U-R

When you run diroll, dxdb creates an Input/Output window. The debugger uses this window to read any user input and to display the appropriate output generated by the program you are debugging.

Figure 5 shows the Input/Output window that dxdb displays when you run diroll.

Figure 5: The Input/Output Window



```
Press <CR> to roll...
+ - - +    + - - +
|     |    | o o |
|  o  |    |     |
|     |    | o o |
+ - - +    + - - +
```

ZK-0108U-R

1.5 Setting Breakpoints and Conditional Breakpoints

You can set breakpoints in your code to pause the execution of the source program. This allows you to examine values. By using a conditional breakpoint, you can specify a condition that will execute at a particular breakpoint.

Breakpoints are represented by a filled-in circle in the Margin Area at the line where you set the breakpoint; the circle also appears in the Breakpoints window.

1.5.1 Setting Breakpoints

When you run diroll, dxdb issues the following error in the Text Area:

```
Bus error in didisp.didisp at line 28 in file "didisp.c"
```

This error indicates a possible indexing problem in the matrices used to build the die in didisp.c. One of several debugging strategies that you can pursue is to set breakpoints at specific areas in the diroll code and then examine the values of specific key variables. These values may provide a lead to the source of the problem.

A good location for the breakpoints is before the call to didisp in diroll.c, and at the printf statement in didisp.c. Setting breakpoints at these positions allows you to observe the values diroll is sending to the function didisp and the effect of those values on the for-loop variables i, j, and k.

You can set breakpoints globally, by line, or by function. In this example, you are setting a breakpoint by line. To set a breakpoint in diroll.c. perform the following steps:

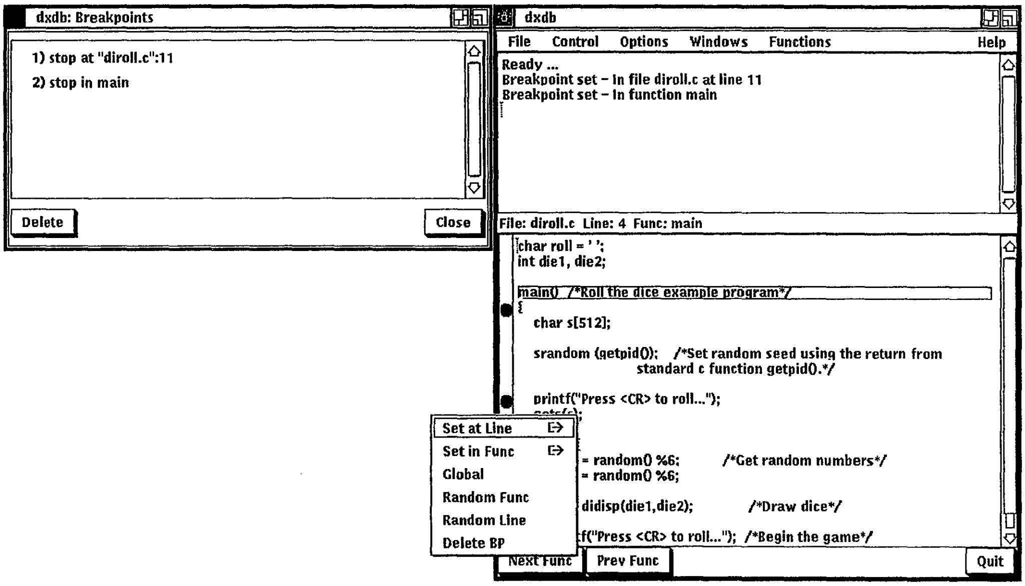
1. Display diroll.c in the Source Area.
2. Move the pointer to the Margin Area at the line containing the call to didisp and press MB2.
3. Move the pointer to the Set at Line option in the Breakpoints menu and release MB2.
4. Repeat this procedure on the printf statement in didisp.c.

Figure 6 shows the Breakpoints window, including the Breakpoints menu and two breakpoints set in the Margin Area.

You can set a random line or random function breakpoint at a line of code not currently displayed in the Source Area by following these steps:

1. Click on the breakpoint symbol using MB2 and choose the Random Function or Random Line option.

Figure 6: The Breakpoints Window with Breakpoints Menu



```
1) stop at "diroll.c":11
2) stop in main
```

```
Ready ...
Breakpoint set - In file diroll.c at line 11
Breakpoint set - In function main
```

File: diroll.c Line: 4 Func: main

```
char roll = ' ';
int die1, die2;

main() /*Roll the dice example program*/
{
   char s[512];

   srandom (getpid()); /*Set random seed using the return from
                        standard c function getpid().*/

   printf("Press <CR> to roll...");

   = random() %6;       /*Get random numbers*/
   = random() %6;

   didisp(die1,die2);       /*Draw dice*/

   f("Press <CR> to roll..."); /*Begin the game*/
```

2. A dialog box appears which prompts you for information about the breakpoint to be set.
3. Click on the OK button.

To cancel a random line or random function breakpoint, click on the Cancel button.

1.5.2 Deleting Breakpoints

There are two ways to delete a breakpoint:

1. Click on the breakpoint symbol using MB2 and choose the Delete BP option of the Breakpoints menu.
2. In the Breakpoints window, click on the breakpoint to be deleted and click on the Delete button. You must use this method to delete global breakpoints that do not have symbols in the Margin Area.

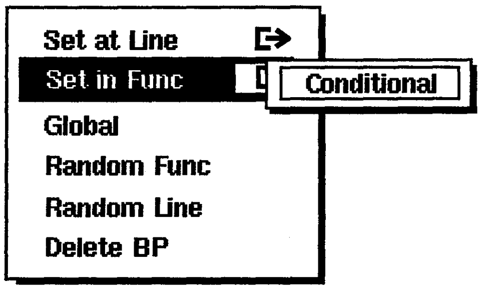
1.5.3 Setting Conditional Breakpoints

You can specify a program condition in which dxdb will execute a particular breakpoint. To set a conditional breakpoint at a line, follow these steps:

1. Move the pointer to the Margin Area at the line where you want to set the breakpoint.
2. Click MB2.
3. Move the pointer to the arrow at the right of the Set at Line option in the Breakpoints menu.
4. Press MB2.
5. Move the pointer to the right and release MB2 with the pointer in the Conditional button. The Condition submenu appears.
6. Enter the condition in the Condition field.
7. Click on the OK button.

To cancel a conditional breakpoint, click on the Cancel button. Figure 7 shows the Breakpoint menu with the Conditional button displayed.

Figure 7: The Breakpoint Menu with Conditional Submenu



1.6 Setting Tracepoints and Conditional Tracepoints

You can also use tracepoints to debug programs. Tracepoints can be set to print the value of a variable at a certain line of code without pausing the execution of your program. Tracepoint information is printed in the Text Area of the Control window. By using a conditional tracepoint, you can specify a condition that will execute at a particular tracepoint.

Tracepoints are represented by a hollow circle in the Margin Area at the line where you set the tracepoint; the circle also appears in the Breakpoints window.

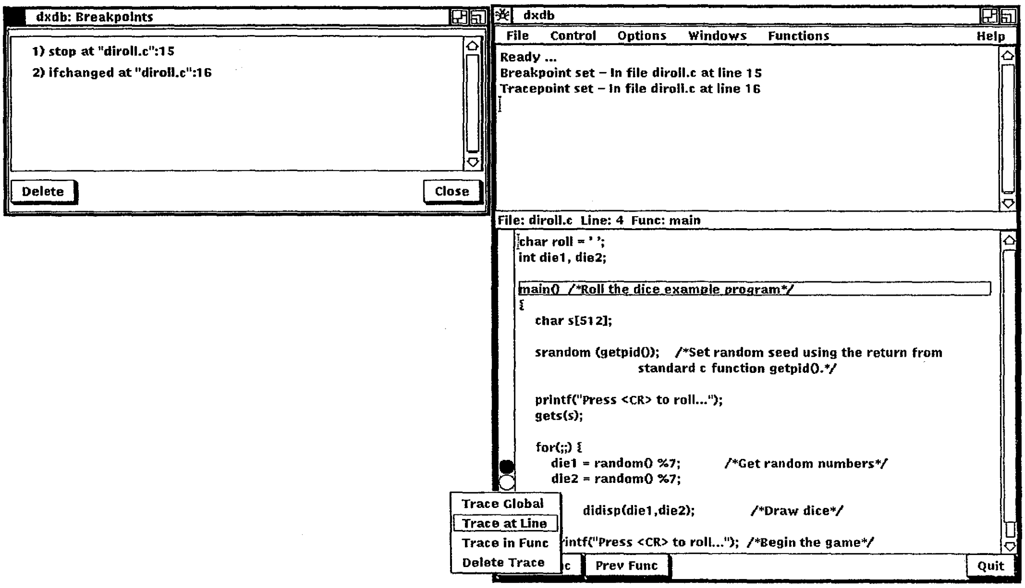
1.6.1 Setting Tracepoints

You can set tracepoints globally, by line, or by function. A good location for a tracepoint in the `diroll` program is immediately after die1 is set. To set a tracepoint in diroll.c. perform the following steps:

1. Display diroll.c in the Text window.
2. Move the pointer to the Margin Area at the line after die1 is set and press shift-MB2.
3. Choose the Trace at Line option.
4. Type die1 into the Trace command field in the Trace box.
5. Click on the OK button.

To cancel a tracepoint, click on the Cancel button. Figure 8 shows the Breakpoints window with the Trace menu.

Figure 8: The Breakpoints Window with Trace Menu



```
1) stop at "diroll.c":15
2) ifchanged at "diroll.c":16
```

```
Ready ...
Breakpoint set – In file diroll.c at line 15
Tracepoint set – In file diroll.c at line 16
```

File: diroll.c Line: 4 Func: main

```
char roll = ' ';
int die1, die2;

main()  /*Roll the dice example program*/
{
   char s[512];

   srandom (getpid());   /*Set random seed using the return from
                      standard c function getpid().*/

   printf("Press <CR> to roll...");
   gets(s);

   for(;;) {
      die1 = random() %7;        /*Get random numbers*/
      die2 = random() %7;

         didisp(die1,die2);         /*Draw dice*/

   intf("Press <CR> to roll...");  /*Begin the game*/
```

1.6.2 Deleting a Tracepoint

To delete a tracepoint, first select the Trace at Line option. There are two ways to delete a tracepoint:

1. Press the shift-MB2 on the tracepoint symbol in the Margin Area and choose the Delete Trace option of the Tracepoint menu.

2. In the Breakpoints window, click on the tracepoint to be deleted and click on the Delete button. You must use this method to delete global tracepoints that do not have symbols in the Margin Area.

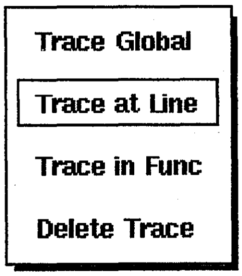
1.6.3 Setting Conditional Tracepoints

You can specify a program condition in which dxdb will execute a particular tracepoint. Perform the following steps:

1. Move the pointer in the Margin Window to the line where you want to set the tracepoint and press shift-MB2.
2. Enter the variable name in the Trace Command field.
3. Enter the condition in the Condition field.
4. Click on the OK button.

Figure 9 shows the Tracepoint menu.

Figure 9: The Tracepoint Menu



1.6.4 Setting Function Breakpoints or Tracepoints

You can set breakpoints if the function appears in the Source Window by selecting the Stop in Func option in the Examine Window (see Figure 10).

You can set tracepoints if the function appears in the Source window by selecting the Trace in Func option in the Tracepoints menu (see Figure 9), or the Examine window (see Figure 10).

1.7 Methods for Selecting Text

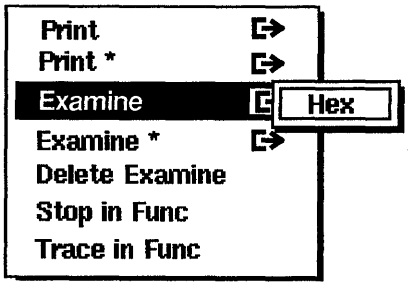
There are two methods for selecting text to be used with the Print, Examine, Stop in Func, and Trace in Func options.

1. Move the pointer to the text you want to operate on and press shift-MB2.
2. Highlight the text you want to operate on.

The option you chose from the Windows menu in the Control window operates on whatever text is under the pointer when the menu is invoked. If there is no text under the cursor, dxdb uses any text that is highlighted.

After you select text, you can print or examine variables. The Print, Print *, Examine, and Examine * options are described in the following sections. Figure 10 shows the Examine menu.

Figure 10: The Examine Menu



ZK-0111U-R

1.7.1 Printing Variables

To print the value of a variable in the Text Area of the Control window, follow these steps:

1. Either move the pointer to the text you want to operate on or highlight the text.
2. Select the variable to be displayed. (See the section on Methods for Selecting Text.)
3. Press shift-MB2.
4. Choose the Print option from the Examine menu.

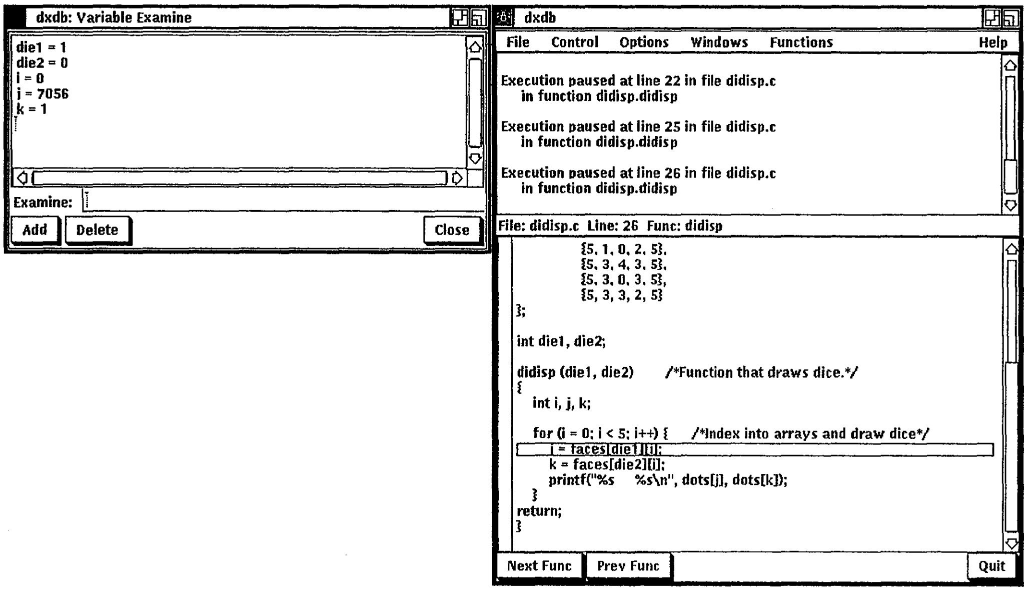
1.7.2 Examining Variables

The debugger provides a separate window for examining the values of various variables (see Figure 11). In the `diroll` example, you will want to examine the variables die1, die2, i, j, and k.

To examine the value of a variable in the Text Area of the Control window, follow these steps:

1. Select the variable to be examined by either moving the pointer to the text or by highlighting the text.
2. Select the Examine option of the Examine menu (see Figure 10). The debugger will display the variable in the Examine window.
3. Repeat steps 1 and 2 on variables die2, i, j, and k.
4. Run the program (see the section on Running a Program earlier in this guide). The debugger displays the current values of all the variables listed in the Examine window each time you stop the program.

Figure 11: The Examine Window



```
die1 = 1
die2 = 0
i = 0
j = 7056
k = 1
```

```
Execution paused at line 22 in file didisp.c
   in function didisp.didisp

Execution paused at line 25 in file didisp.c
   in function didisp.didisp

Execution paused at line 26 in file didisp.c
   in function didisp.didisp
```

File: didisp.c Line: 26 Func: didisp

```
            {5, 1, 0, 2, 5},
            {5, 3, 4, 3, 5},
            {5, 3, 0, 3, 5},
            {5, 3, 3, 2, 5}
};

int die1, die2;

didisp (die1, die2)       /*Function that draws dice.*/
{
   int i, j, k;

   for (i = 0; i < 5; i++) {     /*Index into arrays and draw dice*/
      j = faces[die1][i];
      k = faces[die2][i];
      printf("%s     %s\n", dots[j], dots[k]);
   }
return;
}
```

When you follow these steps, note that if die1 and die2 contain a value equal to 6, the assignment statements in the for-loop (j = faces [die1][i]; or k = faces [die2][i];) attempts to reference the seventh index in the matrix faces. Because there are only six indices in faces, the assignment statement initializes j or k with unpredictable values.

If the value for j or k is unpredictable, then the printf statement also attempts to access an invalid index in the matrix dots. Hence, the printf statement fails and C issues the run-time error.

To correct this error, you would have to edit diroll.c and change the mod values of the random statements from 7 to 6 (%7 to %6). This ensures that random does not generate any numbers greater than 5. (Because C begins a matrix index at zero, zero through 5 provides six indices to a matrix.)

The Examine * option treats the variable selected as a pointer. To examine the displayed values in Hex format, choose the Examine * option and pull right on the arrow and release. Figure 10 shows the Examine menu with the Examine option and the Hex pullright button.

1.7.3 Deleting Variables from the Examine Window

You can delete variables from the Examine window by doing the following:

1. Select the variable to be deleted by either moving the pointer to the text or by highlighting the text.
2. Press shift-MB2.
3. Choose the Delete Examine option.

1.8 Controlling Program Execution

You can examine the values of the variables you listed in the Examine window by executing diroll in single- or multiple-line increments. You set these increments using the Step or Skip window. Display these windows by selecting the Step or Skip options of the Control menu. Figure 4 shows the Control menu.

When dxdb executes a step count, it includes a subroutine call, and the next few lines of that subroutine, as part of the count. When dxdb executes a skip count, it executes the entire routine and resumes the skip count on return from the call. For example, if the step count is 5 and a subroutine call happens to be the second line in the program sequence, dxdb executes the first three lines of the subroutine before pausing.

However, if dxdb performs the same example scenario with a skip count of 5, the debugger executes the entire routine and the next three lines of code immediately following the routine call.

The procedure that follows describes how you use the Step window to set a step count. Note that you use the Skip window to set a skip count in a similar way.

1. Stop the execution of the program by clicking on the Stop menu item in the Control menu (or by setting a breakpoint). Figure 4 shows the Control menu.
2. Click on the Step option.
3. Set the step count to the desired increment. For example, to set the step count to 3, click on the plus (+) accelerator twice and click on the Step Count button.
4. Click on the Step command in the Control menu to step through the code.

As you step or skip through diroll, dxdb updates the Examine Window each time the program changes the variables you have displayed in the examine window. You can then verify that the values of these variables are appropriate for the matrices in didisp.c.

Note that dxdb executes the specified number of lines of code each time you choose the Step or Skip option in the Control menu. If you want to execute a single line at a time, you must return to the appropriate Step or Skip window and either change the count to 1 or click on the Step (or Skip) Once box.

The Continue option runs the program being debugged until a breakpoint is reached or execution ends. The Stop option pauses execution wherever the program is currently running. The Return option finishes execution of the current function and pauses program execution in the line after the function call.

1.9 Assigning a Value to a Variable

At times, you may find it convenient to assign a value to a specific variable. For example, now that you have uncovered the first bug in diroll, you may have noticed that face 6 of the die is slightly deformed. Currently, when didisp.c generates a 6 it displays the following die:

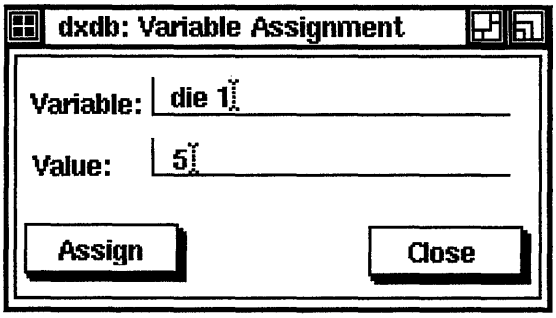
```
+ - - - +
| o   o |
| o   o |
|     o |
+ - - - +
```

The error is caused by the improper organization of values at index 6 of the matrix faces. You can correct this error by editing didisp.c and changing the last row of faces to read {5, 3, 3, 3, 5}.

To verify that your change produces the correct sequence of dots for face 6 of the die, you can assign die1 or die2 the value 5, which will cause didisp.c to index the sixth row of the faces matrix. To assign 5 to die1, perform the following steps:

1. Stop the execution of diroll in didisp.c, before the for statement (by either stepping or setting a breakpoint).
2. Choose Assign from the Windows menu. Figure 12 shows the Assign window.
3. Click on the Variable box of the Assign window and enter die1.
4. Click on the Value box and enter 5.
5. Click on the Assign button.

Figure 12: The Assign Window



ZK-0112U-R

When you have completed these steps, continue running the program. The correct number of dots is displayed for face 6.

1.10 Exiting from dxdb

To end the dxdb session, click on the Quit menu item in the Control Window. When you end a debugger session, dxdb closes all opened files and erases all display windows.

1.11 Additional Tasks

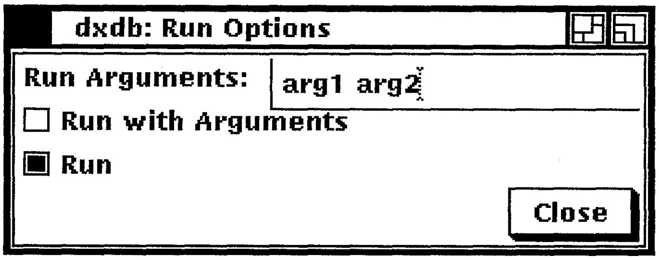
The following sections describe several additional dxdb functions that can facilitate your debugging tasks. These functions allow you to pass parameters to a program to be tested, view the program execution stack, examine all active variables, and get information about variables.

1.11.1 Supplying Arguments to a Program

You use the Run window to supply an argument to a program you are testing. To run a program with an argument, perform the following steps:

1. Choose Run from the Options menu. Figure 13 shows the Run window.
2. Enter the argument you want to pass to the program in the Run Arguments field.
3. Click on the Run with Arguments button.
4. Return to the Control window and run the program.

Figure 13: The Run Window


dxdb: Run Options

Run Arguments: arg1 arg2

☐ Run with Arguments

■ Run

For example, assume you modified the shell command cat and wanted to test the new version on a file called test.cat. To run this test, you would enter test.cat at the Run Arguments prompt, click on the Run With Arguments button, and return to the Control window to run the program.

To run the cat command without arguments, return to the Run window and click on the Run button. When you run cat again, dxdb ignores all the specified arguments.

To close the Run window, click on the Close button.

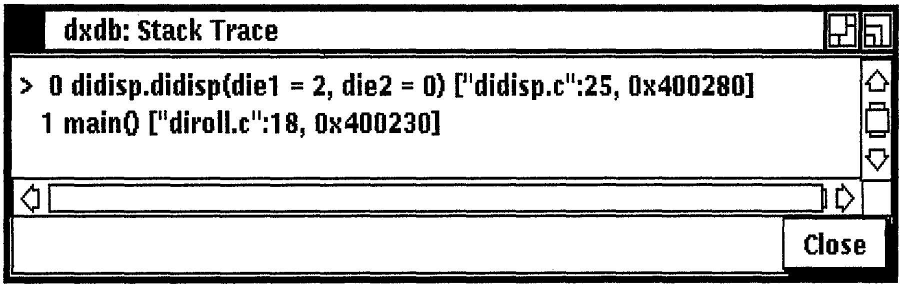
1.11.2 Viewing the Program Execution Stack

The Stack window allows you to view elements of the program execution stack. The program that dxdb is running places an entry on this stack each time the program executes one of its routines. Each stack entry contains the routine name and the parameters passed to that routine.

Using the Stack window, you can trace the exact sequence in which the program called its routines and the parameters the program passed to those routines. This information may help you to uncover any program errors caused by bad parameter passing.

To display the Stack window, choose Stack from the Windows menu. The debugger updates this window (with the current contents of the program execution stack) each time you halt the program. Figure 14 shows the typical format in which dxdb displays this information in the Stack window.

Figure 14: The Stack Window



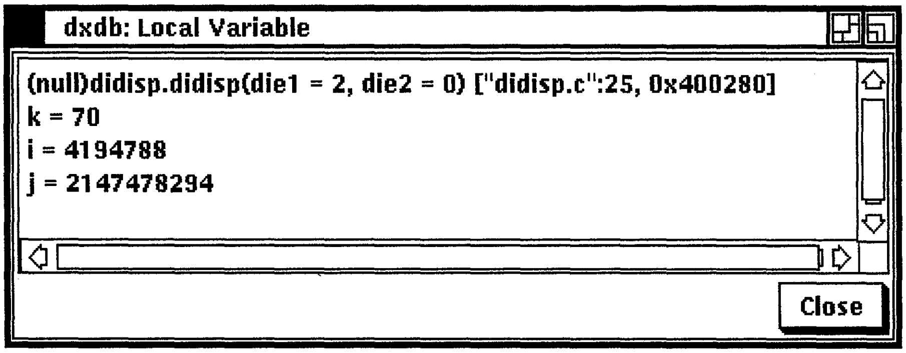
```
dxdb: Stack Trace
> 0 didisp.didisp(die1 = 2, die2 = 0) ["didisp.c":25, 0x400280]
  1 main() ["diroll.c":18, 0x400230]
```

1.11.3 Viewing Variables

The Dump window allows you to view the values of all currently active, local variables. To view all active variables, choose Dump from the Windows menu. Figure 15 shows the Dump window. The debugger updates the contents of this window each time you stop the execution of the program (using the Step, Skip, or Stop function, or by setting a breakpoint).

The Dump window in Figure 15 shows the values of the variables in diroll after the program was stopped in the routine didisp.

Figure 15: The Dump Window



```
dxdb: Local Variable
(null)didisp.didisp(die1 = 2, die2 = 0) ["didisp.c":25, 0x400280]
k = 70
i = 4194788
j = 2147478294
```

1.11.4 Getting Information About Variables

Using the dxdb functions Whatis, Which, and Whereis, you can find out the data type, scope, and location of a variable. The following sections describe each function.

1.11.4.1 Whatis – The Whatis function returns the data type of a variable argument. To find the data type of a variable, chose a variable from the Source window and then choose Whatis from the Functions menu.

The debugger returns the data type of the variable (in the text area of the Control window) in the following format:

```
variable - file.datatype variable;
```

file	The file in which the variable is located
datatype	The data type of the variable
variable	The variable in question

For example, the following message returned by Whatis indicates that the variable **g** is declared an integer in the local routine **tt**:

```
g - tt.int g;
```

1.11.4.2 Which – The Which function returns the scope of the variable (local or global) that is currently active. To use Which, first choose a variable from the Source Window and then choose Which from the Functions menu. Which returns a message in the following format:

```
variable - file.function...variable
```

file	The file in which the variable is declared
function	The function or functions in the routine that use the variable
variable	The variable in question

For example, the following Which message indicates that variable g is used by the function `foo` , which is defined in the file `tt.c` :

```
g - tt.foo.g
```

1.11.4.3 Whereis – The Whereis function returns a list of routines in which a variable appears. To use Whereis, choose a variable from the Source Window and then choose Whereis from the Functions menu. Whereis returns a message in the following format:

```
routine.variable routine.variable ...
```

For example, the following Whereis message indicates that die1 is used in the routines didisp and diroll:

```
didisp.die1 diroll.die1
```

1.11.5 Editing Files

You can edit the file currently displayed in the Source window. Select the Edit option of the File menu. A vi editor session will be started in a DECterm window with the cursor at the current line of execution.

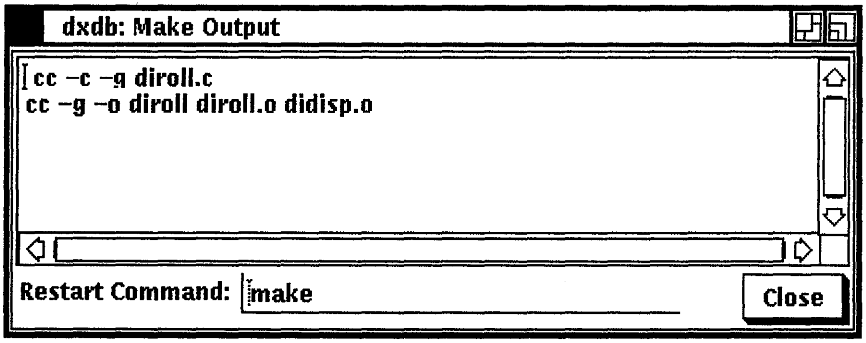
1.11.6 Restarting the Debugger

You can rebuild the executable program you are debugging and restart the debugger using the Make/Restart option of the File menu. This executes the Restart Command that is displayed in the Make window.

To do this, choose the Make Window from the Windows menu. When the Make/Restart command is issued, dxdb executes the text in the Restart command. The default is Make; however, you can change this field to suit the program being debugged. Output from the Restart command appears in the Make Output text window. Figure 16 shows the Make window.

The debugger is restored to its original state. Therefore, you lose any breakpoints or tracepoints you had previously set. You are now running updated source files that are ready to be executed.

Figure 16: The Make Window



```
cc -c -g diroll.c
cc -g -o diroll diroll.o didisp.o
```

Restart Command: make

Index

How to Order Additional Documentation

Technical Support

If you need help deciding which documentation best meets your needs, call 800-343-4040 before placing your electronic, telephone, or direct mail order.

Electronic Orders

To place an order at the Electronic Store, dial 800-DEC-DEMO (800-332-3366) using a 1200- or 2400-baud modem. If you need assistance using the Electronic Store, call 800-DIGITAL (800-344-4825).

Telephone and Direct Mail Orders

Your Location	**Call**	**Contact**
Continental USA, Alaska, or Hawaii	800-DIGITAL	Digital Equipment Corporation P.O. Box CS2008 Nashua, New Hampshire 03061
Puerto Rico	809-754-7575	Local Digital Subsidiary
Canada	800-267-6215	Digital Equipment of Canada Attn: DECdirect Operations KAO2/2 P.O. Box 13000 100 Herzberg Road Kanata, Ontario, Canada K2K 2A6
International	————	Local Digital subsidiary or approved distributor
Internal[1]	————	SSB Order Processing - WMO/E15 *or* Software Supply Business Digital Equipment Corporation Westminster, Massachusetts 01473

* For internal orders, you must submit an Internal Software Order Form (EN-01740-07).

How to Order Additional Documentation

Technical Support

If you need help deciding which documentation best meets your needs, call 800-343-4040 before placing your electronic, telephone, or direct mail order.

Electronic Orders

To place an order at the Electronic Store, dial 800-DEC-DEMO (800-332-3366) using a 1200- or 2400-baud modem. If you need assistance using the Electronic Store, call 800-DIGITAL (800-344-4825).

Telephone and Direct Mail Orders

Your Location	Call	Contact
Continental USA, Alaska, or Hawaii	800-DIGITAL	Digital Equipment Corporation P.O. Box CS2008 Nashua, New Hampshire 03061
Puerto Rico	809-754-7575	Local Digital Subsidiary
Canada	800-267-6215	Digital Equipment of Canada Attn: DECdirect Operations KAO2/2 P.O. Box 13000 100 Herzberg Road Kanata, Ontario, Canada K2K 2A6
International	————	Local Digital subsidiary or approved distributor
Internal[1]	————	SSB Order Processing - WMO/E15 *or* Software Supply Business Digital Equipment Corporation Westminster, Massachusetts 01473

* For internal orders, you must submit an Internal Software Order Form (EN-01740-07).

Reader's Comments

ULTRIX Worksystem Software
Guide to the dxdb Debugger
AA-MA93B-TE

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

Please rate this manual:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

What would you like to see more/less of? ____________________

____

What do you like best about this manual? ____________________

____

What do you like least about this manual? ____________________

____

Please list errors you have found in this manual:

Page	Description

Additional comments or suggestions to improve this manual:

____

____

____

What version of the software described by this manual are you using? __________

Name/Title ____________________ Dept. __________

Company ____________________ Date __________

Mailing Address ____________________

____________________ Email ____________________ Phone __________

Do Not Tear – Fold Here and Tape

digital™

No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

Digital Equipment Corporation
Publications Manager
Open Software Publications Group
ZK03-2/Z04
110 SPIT BROOK ROAD
NASHUA, NH 03062-9987

Do Not Tear – Fold Here and Tape

Cut
Along
Dotted
Line