dbx90: Fortran debugger

March 11, 2024

1 Name

`dbx90` — a Fortran-oriented debugger for use with the NAG Fortran compiler.

2 `dbx90` command line

dbx90 [option]... *executable-file*

3 Description of `dbx90`

dbx90 is a Fortran-oriented debugger for use with the NAG Fortran Compiler on Unix-like systems (e.g. Linux, Solaris). Its syntax is quite similar to that of dbx, which it invokes as a sub-process to carry out the actual machine-dependent debugging commands. (On gcc-based implementations, gdb is used.)

The program to be debugged should be compiled and linked with the *−g90* option. This creates a debug information (`.g90`) file for each Fortran source file.

If the environment variable `DBX90_DBXPATH` is defined, dbx90 will use it to locate the native debugger instead of the built-in path.

4 `dbx90` options

−compatible
: This permits dbx90 to debug code compiled by the NAG Fortran Compiler using the *−compatible* option.

−I ***pathname***
: Add *pathname* to the list of directories which are to be searched for module information (`.mod`) files and debug information (`.g90`) files. The current working directory is always searched first, then any directories named in *−I* options, then the compiler's library directory (usually '`/usr/local/lib/NAG_Fortran`' or '`/opt/NAG_Fortran/lib`')

−Qpath ***path***
: Set the compiler's library directory to *path*. This is only used for finding the module information (`.mod`) files for intrinsic modules.

5 `dbx90` Commands

alias
: List command aliases.

alias ***name text***
: Create a new alias *name* with the replacement value *text*. The replacement text is not limited to a single word but may contain spaces.

assign ***var = expr***
: Assign the value of *expr* to variable *var*. The variable can be scalar, an array (including an array section), a scalar component, or an element of an array component, but cannot be of derived type. The value must be a scalar expression (see "Expressions" for more details).

cont
: Continue execution from where it was stopped (either by a breakpoint or an interrupt).

delete ***n***
: Delete breakpoint number *n*.

delete all Delete all breakpoints.

display List the expressions to be displayed after each breakpoint is reached.

display *expr*
Add *expr* to the list of expressions to display after each breakpoint. *Expr* may also be an array section.

dump Display all local variables and their values.

down [*n*] Move the focus down (i.e. to the procedure called by the current procedure). If n is present, move down n levels.

help [*topic*]
Display a brief help message on the specified *topic*, or on using dbx90 generally.

history List the history command buffer. Old commands can be executed with:

!!	repeat last command,
! *n*	repeat command n in the history buffer, and
! -*n*	repeat the n^{th} last command.

history *n* Set the size of the history command buffer to n commands. The default size of the history command buffer is 20.

if *expr* This is actually a suffix to the breakpoint ('stop') commands not an independent command. It prevents the triggering of the breakpoint until the expression *expr* (a scalar expression) is `.TRUE.` (or non-zero).

list [*line1*[,*line2*]]
Display the next 10 lines of the program, display line *line1* of the current file or display lines *line1* to *line2* of the current file.

next [*n*] Execute the next n lines (default 1) of the current (or parent) procedure. Any procedure reference in these lines will be executed in its entirety unless a breakpoint is contained therein.

print *expr* [,*expr*]...
Display the value of *expr*, which can be a scalar expression, array section, derived type component,or a variable of any data type. Several expressions may be given separated by commas.

quit Exit from dbx90, immediately terminating any program being debugged.

raw *dbx-command*
Pass *dbx-command* directly to "dbx". This is not recommended for normal use.

rerun [*command-line*]
Begin a new execution of the program, passing *command-line* to it (if present) or the *command-line* from the previous run or rerun command if not.

run [*command-line*]
Begin a new execution of the program, passing *command-line* to it (if present) or blank command line if not.

scope [*name*]
Display the current procedure name or set the focus to the specified procedure *name*.

status List the breakpoints which are currently set.

step [*n*] Execute the next n lines (default 1) of the program, counting lines in referenced procedures (i.e. step into procedure references).

stop *name*
Set a breakpoint which triggers when variable *name* is accessed. Note that *name* cannot be '`at`' or '`in`'. This command is not available on Solaris or HP-UX.

stop at *lineno*
Set a breakpoint at line *lineno* of the file containing the current procedure.

stop in *name*
Set a breakpoint at the beginning of procedure *name*. Note that this breakpoint occurs at the beginning of procedure initialisation, not at the first executable statement. If there is no procedure called 'MAIN', the main program can be specified using that name.

undisplay *expr*
Remove *expr* from the "display" list.

up [*n*] Move the focus up (i.e. to the caller of the current procedure). If *n* is present, move up *n* levels.

whatis *name*
Describe how *name* would be explicitly declared.

where Display the stack of active procedures with their dummy argument names.

which *name*
Display the fully qualified form of *name* which can be used for access from another scope.

6 dbx90 Expressions

6.1 Scalar expressions

Scalar expressions in dbx90 are composed of literal constants, scalar variable references including array elements, intrinsic operations and parentheses.

Literal constants can be of any intrinsic type, e.g.

```
INTEGER    42
REAL       1.2
           1.3e2
COMPLEX    (5.2,6.3)
CHARACTER  "string"
LOGICAL    .TRUE.
           .T.
```

Subscript expressions must be scalar and of type `INTEGER`.

All intrinsic operations are supported except for exponentiation and concatenation, that is:

`+`, `-`, `*`, `/`, `==`, `/=`, `<`, `<=`, `>`, `>=`, `.AND.`, `.OR.`, `.NOT.`, `.EQV.`, `.NEQV.`, `.EQ.`, `.NE.`, `.LT.`, `.LE.`, `.GT.`, `.GE.`

(Operator names are not case-sensitive).

Note: array operations and operations involving variables of complex, character or derived type are not supported.

6.2 Array sections

Assignment, printing and displaying of array sections follows the Fortran syntax, e.g.

```
ARRAY(:)
ARRAY(1:5)
ARRAY(1:10:2)
```

If the stride is supplied it must be a positive scalar expression – negative strides are not supported. All subscript expressions must be scalar – vector subscripts are not supported.

6.3 Derived type component specification

Individual components of a derived type scalar or array may be printed using normal Fortran syntax.

For example,

```
print var%a
```

will print the "a" component of derived type "var".

Components of all data types are supported.

Components which are of derived type will be displayed recursively until either:

a. there are no further nested derived types, or

b. a derived type array component is reached.

Array components of intrinsic data types will be truncated to '`<array>`', and derived type array components will be truncated to '`<derived type array>`'.

Allocatable components of derived types are supported.

Derived type assignment is not supported; however, scalar non-derived-type components may be assigned values.

7 dbx90 Command aliases

The following set of command aliases are defined:

```
a   assign
b   stop at
bp  stop in
c   cont
h   history
l   list
n   next
p   print
q   quit
r   rerun
s   step
```

New aliases may be created using the `alias` command, e.g.

```
alias xp1 print x+1
```

8 dbx90 limitations

Breakpoints set at the beginning of a routine occur before procedure initialisation; at this point attempting to print or display an assumed-shape dummy argument, a variable with a complicated `EQUIVALENCE` or an automatic variable will produce dbx crashes, dbx90 crashes or nonsensical output. Execution must be stepped to the first executable statement (e.g. using the `next` command or by setting a second breakpoint) before any of these will work satisfactorily.

Strides in array sections must be positive.

9 Example of dbx90

This is an example of the use of dbx90 in debugging some Fortran code which contains both `COMMON` blocks and modules.

The file to be debugged is called '`fh4.f90`' and contains:

```
MODULE fh4
  REAL r
```

```
END MODULE fh4

PROGRAM fh4_prog
  USE fh4
  COMMON/fh4com/i
  i = 2
  CALL sub
  PRINT *,i,r
END PROGRAM fh4_prog

SUBROUTINE sub
  USE fh4
  COMMON/fh4com/i
  r = 0.5*i
  i = i*3
END SUBROUTINE sub
```

It is first compiled with the *−g90* option and then run under dbx90:

```
% nagfor -g90 -o fh4 fh4.f90
% dbx90 fh4
NAG dbx90 Version 5.2(22)
Copyright 1995-2008 The Numerical Algorithms Group Ltd., Oxford, U.K.
GNU gdb Red Hat Linux (6.5-15.fc6rh)
Copyright (C) 2006 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i386-redhat-linux-gnu"...Using host libthread_db lib
rary "/lib/libthread_db.so.1".

(dbx90)
```

Setting a breakpoint in routine `SUB` and running the program.

```
(dbx90) stop in sub
[1] stop in SUB in file "fh4.f90"
(dbx90) run
stopped in SUB at line 16 in file "fh4.f90"
   16     r = 0.5*i
(dbx90)
```

Printing the value of a variable, which may be local, in a `COMMON` block, or in a `USE`d module.

```
(dbx90) print i
I = 2
(dbx90) next
17        i = i*3
(dbx90) print r
R = 1
(dbx90) next
18      END SUBROUTINE sub
(dbx90) print i
I = 6
```

Variables can also be assigned values.

```
(dbx90) assign i = 7
I  = 7
(dbx90) cont
 7   1.0000000

Program exited normally.
(dbx90) quit
%
```

10 Troubleshooting dbx90

The diagnostic messages produced by dbx90 itself are intended to be self-explanatory.

If you receive the error message '`Cannot exec dbx`' when starting dbx90 then you must set the environment variable `DBX90_DBXPATH` to the pathname of dbx (or gdb, or xdb).

11 See Also

dbx(1), **gdb(1)**, **nagfor(1)**.

12 Bugs

It is called dbx90 because it was first available for Fortran 90; that name is no longer appropriate. Please report any other bugs found to 'support@nag.co.uk' or 'support@nag.com', along with any suggestions for improvements.