MULTICS TECHNICAL BULLETIN

To: MTB Distribution

From: Jim Davis

Date: 06/06/79

Subject: The future of probe

I Introduction

This MTB describes the plans for probe in Multics Release 8.0 and beyond. The most important feature of version 4 probe (in MR 8) will be improved support of FORTRAN and COBOL. Other features will be support for low-level debugging, self-documentation, and several convenient requests. The implementation will be drastically changed to allow for future expansion. This MTB concludes with ideas for future versions of probe, and is followed by a revised MPM document.

II Improved Support of FORTRAN and COBOL.

The current probe user interface was designed with the PL/I user in mind, and can be annoyingly inconvenient for COBOL or FORTRAN users. The goal is to present each programmer with a "virtual machine" that seems to work in the programmer's language. To this end, probe will have a "current language", which will determine how expressions are interpreted and how to format output.

One problem with the current design of probe is that the source language might conceivably be specified by either the "source" pointer or the "block" pointer. Since neither is the obvious right choice the two will be merged into a single entity, which will subsume the functions of both, although it will continue to be called the "source" pointer.

When the current language is COBOL, probe will:

translate upper case variable names to lower case, and translate the hyphen character to an underscore on input. Output will use the name as stored in the symbol table, except that underscore will be translated to hyphen. This translation is necessary because the COBOL compiler use all lower case letters, and substitutes the underscore for the hyphen in symbol tables.

Use COBOL, rather than PL/I, terms when printing the attributes of data.

Use the COBOL structure name qualification syntax instead of the PL/I syntax.

Probe (and debugger support routines in bound_debug_util_) will be extended to handle the COBOL data types for 4 bit decimal and overpunched data. (Note that COBOL currently does not build correct symbol table entries for some of these data types).

In addition, probe will refuse to set breaks *after* COBOL statements. Because of the code generated, this feature was never reliable, so it will be removed.

When the current language is FORTRAN probe will:

accept the FORTRAN comparison operators in addition to the PL/I operators.

use .true. and .false. for input and output of logical data, instead of "1"b and "0"b.

ignore case on input if the FORTRAN object segment was compiled with -fold or -card. On output the names of variables will be printed as they appear in the symbol table.

In addition, the new request "display" will be useful to the FORTRAN programmer who has stored Multics pointers or other non-FORTRAN data in FORTRAN storage.

III Lower Level Support

Lower level support is being added to probe in order that debug may be phased out. It is undesirable to maintain two debuggers, and probe is far more maintainable than debug. There are no plans now to do anything to debug. It is hoped that it will simply wither away. New features that are applicable to low level debugging are:

The new request "display", mentioned above.

Machine registers (as saved in the stack frame) will be accessible as probe builtins.

IV Extensions to probe

third goal is to make probe more easily extensible. Extensions are contemplated for such features as debugging dead processes and other peoples live ones, and supporting PASCAL. Although no extensions are planned for release 8.0, all internal architecture changes to support them will be available. In fact, the support of FORTRAN and COBOL depends upon it.

V Other New Features

Probe will be self-documenting. Two new requests will be added: "help", which prints info files, and "?", which prints the names of all currenly defined requests.

Several new builtins will be added, including addrel, baseptr, rel and ptr. A full list is in the MPM writeup.

Extensive control over probe's behavior will be available. Instead of just one mode ("long" or "brief") there will be many orthogonal modes, with separate control of features that were previously all controlled by "mode". Modes will be saved across process, in the users probe seg.

It will be possible to print all the arguments to the current procedure with a single request.

The syntax of the "status" and "reset" requests will be simplified and made consistent with each other.

The search string supplied to the "position" request is stored, so the user can search for identical string by supplying "".

Expressions will be evaluated using operator precedence. (Currently they are evaluated in a left-to-right manner)

VI Documentation

Enclosed is a completely new probe MPM write-up. There is no glossary or examples in it - the 7.0 versions will be used. It has not been determined whether FORTRAN or COBOL examples will go into the MPM (as opposed to the FUG and CUG).

VII Wishful Thinking

There has been some discussion of features that could/should be added to probe "someday". They are currently not well defined, but are included here for speculation.

editing breakpoints using qedx_

a "smart" step request that uses an extended statement map so that transfers of control (if-then-else, do loops) no longer result in stepping getting "lost".

the ability to trace all references to a variable, and execute a breakpoint when a given location is either read or written.

the ability to specify source line location using non-executable source lines such as comments

The ability to search using regular expressions.

The ability to set breakpoints at arbitrary locations, not just the first or last instructions of a statement.

Extending conditional expressions to use the Boolean operators (and, or, not).

Completing the implementation of macros. There is code in probe now to process macros, which are like breakpoint request lists, but can be given names and executed by the request "macro foo".

A "video" probe - with real-time updating of stack trace and variable values when breakpoints occured, and coordinated display of source text.

Name: probe, pb

The probe command provides symbolic, interactive debugging facilities for programs compiled with PL/I. FORTRAN, or COBOL. Its features permit a user to interrupt a running program at a particular statement, examine and modify program variables in their initial state or during execution, examine the stack of block invocations, and list portions of the source program. External subroutines and functions may be invoked, with arguments as required, for execution under probe control. The probe command may be called recursively.

Usage

```
     probe {procedure_name}
```

where procedure name is an optional argument that gives the pathname or reference name of an entry to the procedure or subroutine that is to be examined with probe.

Overview of Processing

When probe has been invoked it accepts requests from the user. A probe request consists of a keyword (or its abbreviation) that specifies the desired function and any arguments required by the particular request. Requests are separated from each other by newlines or semi-colons.

A series of requests may be given in the form of a request list. This is used in breakpoint request lists and conditional execution lists. Here, each request is separated by semicolons. An example:

```
   (value a; v b; continue)
```

Probe at all times has a "current language". It communicates with the user in terms appropriate to the language of the procedure being examined. The syntax of an expression and the form of probes output vary from language to language.

To use probe to the fullest, a program must be compiled so that the object segment produced has both a symbol table and a statement map (these terms, and others, are defined below in the Glossary). A symbol table and statement map are produced for the languages supported if the -table control argument is given to the compiler. A program may also be compiled with the -brief_table control argument, which produces only a statement map. In this case the user may retrieve information about source statements and where the program was interrupted, and may set breakpoints, but can do little else.

Probe Pointers

Two internal "pointers" are used by probe to keep track of the program's state. They are the "source" pointer and the "control" pointer.

The source pointer identifies a line, a block, and a frame. A line is a source program line number. The language of the source line is the language probe will use with the user. The meaning of a block depends on the language. For a PL/I program, it specifies the smallest begin block or procedure that contains it. For a FORTRAN program it specifies the program or subprogram the statement occurs in. For a COBOL program it indicates the program-id of the containing program. The frame specifies a stack frame associated with the block. When there are several invocations of the same block on the stack, the frame distinguishes between them. If there is no activation of the block, then the frame portion of the source pointer is null. In this case, certain types of storage (i.e. PL/I automatic) are not defined. The initial value of the source pointer is determined by the initial value of the control pointer.

The control pointer indicates the last location executed befor probe was invoked. The initial value depends on the manner probe was invoked.

1) If probe is invoked from a breakpoint, then the control pointer is set to the line where the break occurred.
2) if probe was invoked from the command line, then if a procedure_name was specified, then if the procedure is active, then the control pointer is set to the last line executed in the most recent invocation of that procedure.
3) if the procedure in the command line was not active, then the control pointer is set to the entry statement for the procedure.
4) If no procedure_name was specified, then if there is a QUIT signal or condition frame on the stack, then the control pointer is set to the location being executed when the condition was signalled.
5) If no procedure_name was specified, and there are no condition frames on the stack, then the last line executed in the the most recent frame will be used. (This will usually be the command processor).

Information about programs being debugged is stored by probe in a segment in the user's home directory called Person_id.probe, where Person_id is the user's log-in name. This segment is created automatically when needed. This segment should not be deleted, or probe will be unable to reset any breaks it has set.

Restrictions on input lines:

If a probe input line contains unbalanced parenthesis or quotes, the user is warned. This means that a request or request list as typed in must fit on one line. It cannot contain a newline character. If a long line must be typed the Multics escape convention of placing a backslash before the newline may be used. If the newline character is needed (in a character string constant, for example), then the escape sequence \012 may be entered instead.

PROBE REQUESTS

The following pages present the format and function of each probe request, giving first the name of the request, and its abbreviated form, if any, and its arguments, required and optional. The syntax of the arguments is described in the following way:

- brackets enclose optional material
- where the user may select only one of several options, curly braces enclose the list of choices, and the choices are separated by a vertical bar:
- upper case names represent items whose syntax is elsewhere defined (for example, EXPRESSION or PLACE)

The following items are used throughout the requests section:

N, M
: are positive, unsigned integers

PROCEDURE
: is a path name or reference name of a procedure

PROCEDURES
: is a list of zero or more PROCEDUREs, separated by space

REQUEST
: is any probe request (or requests)

PATH
: is a Multics pathname.

LINE
: is a line of program text in a source segment and/or the set of instructions in the object segment corresponding to that text. It is defined below in THE SYNTAX OF A LINE.

EXPRESSION
is an expression, defined below in THE SYNTAX OF AN EXPRESSION

STRING
a quoted string. Its beginning and end are delimited by ". If a quote character is to be included in the string, two quotes should be used. (i.e. "this is a quote: "" character")

Examples:

```
request ARG1 [OPT ARG] [-reset]  [[-long | -brief]]
```

This request must be supplied with a value for ARG1, and may be followed by OPT ARG or "-reset", and either "-long" or "-brief". ARG1 and OPT ARG should be defined by the writeup for this request

BASIC REQUESTS

0 .

This request causes probe to identify itself by printing "probe" and the current version number on the terminal. It may be used, for example, to determine if a called routine has returned. The version number is useful for determining whether the version of probe being used has certain features or bug-fixes. It should always be included in any trouble report about probe.

0 ..
.. COMMAND LINE

This request passes the remainder of the line (COMMAND LINE) directly to the Multics command processor. It can never be used in a break request list or a conditional execution list. When used, it must be the first request on the line.

Example:

```
..wd; ls *.pl1
```

```
O value, v
  value { EXPRESSION | CROSS-SECTION }
```

The value of the given EXPRESSION or in the elements of the array specified by the CROSS-SECTION are displayed on the user's terminal.

A CROSS-SECTION is specified by giving the upper and lower bounds of one or more subscripts. An asterisk may be used, which is equivalent to a cross-section from the lowest to highest subscript of an array.

Examples:

```
value arr (1:5, 3:7)
value p -> a.b(1)
value a of b of lrec
value ijptr(*,3)
```

The value request may be used with PL/I structures or COBOL records, in which case the value of every component is displayed as well.

The value request cannot be used with PL/I areas.

External functions may be called with the value request. The argument list may involve arbitrary expressions, and the arguments will be converted to the proper type, if the called function specifies what type of arguments are expected.

```
O let, l
  let { VARIABLE | CROSS-SECTION } = EXPRESSION
```

This request sets the specified variable or array elements to the value of the expression. If the variable and expression are of different types, conversion is performed according to the rules of PL/I. Array cross-sections are expressed as in the value request. One array cross-section may not be assigned to another, nor may structures be assigned to as a whole. Certain PL/I data types may only assigned to identical types. For example, areas may only be assigned to areas, and files may only be assigned to files.

Note that because of unpredictable compiler optimization, the change may not take effect immediately, even though the value request shows that the variable has been altered.

O help
  help [[TOPIC | *]]

The help request, invoked with no argument, or invoked with the argument "*", prints a list of all available topics. If it is invoked with an argument that is not "*", it prints information about TOPIC, if there is any.

Examples:

```
 help
 help expressions
```

O quit, q
  quit

This request causes the current level of probe to return. If there is more than one invocation of probe on the stack, the user may still be in probe. If there is only one, then this request causes a return to command level.

SOURCE REQUESTS

The source pointer is used to indicate a block in a program (to resolve variable name conflicts) and a stack frame (to resolve separate invokations of a block), and a statement, (to be printed).

Its current value may be displayed with the "where" request, its value may be changed by the "position" or "use" request. The source line pointed to may be printed via the "source" request.

O where
  where [[sc | ctl | source | control]]

The where request displays the values of the probe pointers. If it is invoked with no argument it displays the values of both, otherwise it displays the value of the pointer named.

O use, position, ps
  use [[LINE | +N | -N | PROCEDURE | [<] STRING]]

This request may be invoked by either of its names (position or use). If invoked as position the line positioned to is displayed. If invoked as use then there is no display.

If no argument is supplied the source pointer is reset to its initial value, which is the value of the control pointer.

The new value of the source pointer can be given in a variety of ways:

- absolutely, by giving the LINE within the current procedure.
- relatively, by giving +N or -N. The new source pointer value is the statement N statements after or before the current statement.
- a new stack frame may be specified by level N, where N is the number of the stack frame of interest. If too high a number is given, the highest numbered frame is used.
- a new procedure may be specified by giving its path or reference name (PROCEDURE).
- by searching: the user can request that probe search through a source segment for an executable line containing STRING. An empty quoted string ("") causes probe to use the last search string. It is an error to use an empty quoted string expression if there has not been a previous quoted string. A less-than character causes the search to be done in reverse. If the searching fails, then the source pointer is not changed.

Examples:

```
    use level 4
         specify level number - last line executed at level 4
    ps -4
         statement four statements before current one.
    ps 2-34
         include file 2, line 34, current procedure
    ps label
         set to line whose label is "label"
    ps "label:"
         search for line containing the word "label" followed
         by a colon. In effect, the same as the previous
         example.
```

Note that probe deals with executable statements, not source lines. The source pointer cannot be set to a source line for which no instructions are executed. This includes blank lines, comment lines, declarations, and COBOL declarative procedures. It is not possible to search for non-executable lines either.

O source, sc
  source [{ N | path PATH }]

The source request displays lines of the source program, beginning with the one pointed to by the source pointer. It never alters the source pointer. If no argument is supplied, one line is displayed, otherwise N are.

A statement can take up many lines, and there may be blank lines (or non-executable source lines, such as comments or declarations) between statements. Although the source pointer can only be set to a line for which code is generated, these lines may be displayed along with the statements. If N statements are displayed, any non-executable lines between the first and the last will also be displayed.

Multics object segments contain within them the absolute pathnames of the source segments used to compile them. Sometimes these segments have been moved by the time the object segment is being debugged, and when probe attempts to locate them, it will fail. When it does, it informs the user that the source cannot be located, and the user can supply probe with the path of the source by giving it after the path argument.

BREAK REQUESTS

It is possible to insert a breakpoint either before or after any statement for which object code was generated. When a transfer is made to statement x, a break before statement x is effective, but a break set after statement x-1 is not effective. A break set after a statement that transfers control (such as a goto or return) may not be executed.

No breaks may be set after any COBOL statement.

The syntax to set either kind of break is the same:

```
O before [LINE] [: REQUEST ]
O after  [LINE] [: REQUEST ]
```

LINE indicates some statement where the break is to be inserted. If none is supplied, the statement identified by the source pointer is used. A set of probe requests may be associated with the breakpoint by placing the requests after a colon. If no requests are given, then "halt" is used.

Examples:

```
b 2
after foo,2: if a > 7:halt
b 52: (v f; v j; v c; continue)
```

O status, st
  status [MOD] [LINE] [{ -all | PROCEDURE}] [-long]

The status request lists the breaks set by probe in various procedures. If LINE is not specified, then all lines in the selected procedure are listed, otherwise one line is listed. MOD is used to distinguish the break before LINE from one after LINE, if necessary. It may be "before", "b", "after", "a", or "at". If it is not given, then the status of breaks both before and after is displayed. If PROCEDURE is not specified, then the current procedure is used, unless -all is specified, in which case all procedures known to probe are used. If -long (abbreviated as -lg) appears, then the break request list associated with the breakpoint is printed.

Examples:

```
     status >udd>Dog20>Crecho>zlotny      lists   all   breaks   in
                                          zlotny
     status 35 -lg                        lists  breaks  before  or
                                          after line 35
     st b 7                               lists  only   the   break
                                          before line 7
     st -all                              lists all breaks  in  the
                                          world
```

O reset, r
  reset [MOD] [LINE] [{-all | PROCEDURE}] [-brief]

This request resets breakpoints set by probe at selected lines in selected procedures. If LINE is not specified, then all breaks in the selected procedure are reset. MOD is defined just as for the "status" request. If PROCEDURE is not specified, then LINE applies to the current procedure, unless -all is given, in which case LINE applies to all procedures known to probe. As breaks are rest, the source line number and pathname of the containing segment are printed, unless -brief is given, in which case nothing is printed.

Examples:

```
   r                             Reset the break at the current line
   r -all                        Reset every break probe can find
   r a $259,1                    Reset the  break  after  the  first
                                 statement  after  the line labelled
                                 "259"
   r -nf                         Reset all  breaks  in  the  current
                                 procedure
   r 259 <Weir>estimate_prophet  Reset  breaks  at  line  259  in
```

segment of given path

REQUESTS USEFUL IN BREAKPOINT REQUEST LISTS

O halt, h
  halt

This causes probe to stop processing the request list and read requests from the terminal. A new invocation of probe is created, with the control and source pointers set to the line of the breakpoint. After a subsequent continuation probe will resume interpreting the break request list that contains the halt. When the list is empty, the users program is resumed. This request has no effect when issued from the terminal.

Example:

halt	this breakpoint request list stops
(v a; halt; v a)	this list displays the value of a before and after stopping

O pause, pa
  pause

This request is equivalent to "halt;reset" in a break request list. It causes the procedure to execute a breakpoint once, and then reset it when execution is resumed. It has no effect if not executed in a breakpoint request list. If the user does not eventually continue the breakpoint then the break will not be reset.

FLOW OF CONTROL REQUESTS

Requests are provided for selected execution of program statements. The user can resume execution after a break, call external procedures, or perform explicit "goto"s.

O continue, c
  continue

This request restarts a program that has been suspended by a probe breakpoint. If this request is used in any other context, probe returns to its caller, which is usually command level.

```
O step, s
  step
```

This request attempts to step through the current program one statement at a time. If the program has been stopped before line N, a break is set before line N+1. If the user is stopped after line N, the break is set before line N+2. These breaks contain "pause" as their sole request list, and thus are self-reseting. If the statements being stepped do not execute in sequence, then the stepping may be unsucesful. Note that PL/I and FORTRAN do-loops, and conditional statements in all languages, do not execute sequentially.

```
O continue_to, ct
  continue_to LINE
```

This request inserts a temporary breakpoint before the LINE specified, then continues. The effect is as is the user had typed the following:

```
  before LINE: pause
  continue
```

Example:

```
  ct 1
```

```
O call, cl
  call PROCEDURE (ARGUMENTS)
```

This request calls the external procedure named with the arguments given. PROCEDURE should be the pathname or the reference name of a program. ARGUMENTS should be a list of arguments to the called procedure, separated by commas. If the procedure expects arguments of a certain type, those given are converted to the expected type. The value request (see above) can be used to invoke a function, with the same sort of conversion occuring. If the procedure has no arguments, an empty argument list "()" must be given.

Examples:

```
     call sub ("abcd", p -> p2 -> bv, 25b, addr(k))
     call eat-master (a of b of new-unit, REC-LEVEL)
```

0 goto, g
  goto LINE

This request transfers control from probe to the statement specified and initiates execution at that point. The syntax of a LINE is given below. It is an error to use this request to goto a line in a a procedure that is not active. Because of compiler optomization, it can be dangerous to use this request.

Examples:

goto label_var	transfer to value of label variables
goto action (4)	transfer to value of label constant
goto 110	transfer to statement on line 110
goto $110	transfer to line with label 110
goto $c,1	transfer to the statement after the current one

CONDITIONAL PREDICATES

Probe provides two forms of conditional execution. The "if" request evaluates a conditional expression, and executes a request list if the expression is true. The "while" request repeatedly executes a request list, testing the conditional expression before each execution. The format of a conditional expression is:

 EXPRESSION OP EXPRESSION

where OP can be <=, <, =, ^=, > or >=. When the current language is FORTRAN, .le., .lt., .eq., .ne., .gt., .ge. are also accepted.

0 if
  if CONDITIONAL EXPRESSION : REQUESTS

This request is most useful in a break request, where it can be used to cause a conditional halt. REQUEST may be a single request, or several probe requests, enclosed in parenthesis and separated by semi-colons.

```
0 while, wl
  while CONDITIONAL EXPRESSION : REQUESTS
```

Examples:

```
  if a < b: let p = addr(a)
  while p ^= null: (v p -> r.val; let p = p -> r.next)
  if ijk .ne. 8: halt
```

REQUESTS TO CONTROL PROBE

It is possible to control probes behavior in a few ways - the length of error messages, the amount of printing done by breaks and by the value request can all be controlled. The current language can be specified explicitly. In addition, the streams used by probe for input and output can be controlled.

```
0 modes, mode
  modes [MODES]
```

The modes request sets various mode internal to probe which change the way it functions. If no arguments are given, the current modes are printed. MODES may be any combination of the following. If conflicting modes are set, the last one in the request determines the setting of the mode. Modes are stored in a segment in the user's home directory named Person_id.probe, and thus remain set across processes. In the description below, LEN should be either "long" ("lg"), "short" ("sh"), or "brief" ("bf"), and is used to specify the kind or amount of printing to be done by a given part of probe. The amount of output produced is greatest for "long" and least for "brief", with "short" in between. In some cases, "short" and "brief" will be the same.

error_messages, em LEN
: controls the length of the text used for an error message. The default is long

qualification, qf LEN
: controls the way variable names are printed by the value request. The default is "brief", which causes only the last name of a structure to be printed. If it is "long", then names are printed fully qualified. This mode only affects the printing of PL/I names.

value_print, vp LEN
: Controls the circumstances under which the value request prints the name of a variable. The default is "short" which

prints the name only for structures or arrays. If it is "long" then the name is always printed, and if it is "brief" then the name is never printed.

value_separator STRING

Causes the value request to print STRING between the name of a variable and its value, if the name is being printed. Only the first 32 characters of STRING are used. The default is " ".

exclude STARNAMES

Where STARNAMES is a sequence of one or more names that obey the star convention, causes the value request to ignore any names in structures or records that match the name. Each use of this control argument resets the entire list of ignored strings to the ones specified. To reset the list to ignore no elements, use "modes ignore """. The default is to ignore nothing. This mode must be the last in the requesty, for all names appearing after it are treated as names to ignore.

The Multics command probe_modes can be used to set any of these modes without invoking probe. It is useful in a start_up.ec. It accepts precisly the same syntax as the modes request.

O input_switch, isw
  isw [SWITCH]

This request causes probe to take all further command input from the switch named. If no SWITCH is supplied, then user_input is used. If there are any other requests in the input line or break request list that contains this request they will be ignored without comment. Input is read from the switch until either a new input_switch request is read, or all available characters are processed, in which case a message is printed and input is reset to user_input. If any errors occur input is reset to user_input. The switch SWITCH must be attached and open before this request is given.

O output_switch, osw
  osw [SWITCH]

This request causes probe to direct all its output to the switch named. If SWITCH is not specified, user_output is used.

O language, lng
  language [LANG]

If no argument is given, this request prints the name of the "current langauge". Otherwise LANG should be the name of one of the supported probe languages. Names accepted are: PL/I, pl1, PL1, FORTRAN, fortran, FT, ft, COBOL, and cobol.

O display, ds
  display [*] VARIABLE [FORMAT] [N]

The display request displays an arbitrary location in a selected format. If an asterisk appears before VARIABLE, then indirection is specified, and the value of the variable specifies the address of the storage to be displayed, otherwise the address of VARIABLE is the address of the first location displayed. It is an error to use display with a VARIABLE that has no storage (such as a format constant) or with a literal constant, unless indirection is used, in which case VARIABLE may be an addressing constant (such as label constant), a pointer constant, or an expression with a pointer result.

FORMAT may be one of the following:

-octal, -o
   is the number of (36 bit) words dumped.
-ascii, -character, -ch
   N is the number of characters dumped. A non-printable character is printed as "."
-instruction, -i
   N is the number of instructions dumped. If the instruction has descriptors, they are dumped with the instruction.
-pointer, -ptr, -its
   N is the number of ITS pointers displayed.

The default FORMAT is octal, and the default for N is 1.

Examples:

ds * 253|100 -octal 20	dumps 20 words in octal
ds foo -ascii 64	displays the first 64 characters of foo

O stack, sk
  sk [[M ,] N] [all]

This request traces the stack backwards and displays the first N frames. If M is not given then the highest numbered frame is the first frame displayed, otherwise the M'th frame is the first displayed. If N is not given then all frames are displayed, and M cannot be specified. System support frames are not displayed unless "all" is given.

For each block, the frame number is given, as is the name of any condition raised in the block.

Examples:

stack	traces the whole stack
stack 2	displays the two most recent frames
stack 3,2	displays two frames starting with frame 3

```
0 arg
  args
```

The args request displays the names and values of the arguments to the current procedure.

```
0 symbol, sb
  symbol VARIABLE [long]
```

This request displays the attributes of the variable specified and the name of the block in which it is declared. If the size or dimensions of the variable are not constant an attempt is made to evaluate the size or extent expression; if the value cannot be determined an asterisk (*) is displayed instead. If "long" appears after the name of the identifier, and if the identifier is a PL/I structure or COBOL record, then the attributes of all members of the structure or record are displayed as well.

```
0 execute, e
  execute STRING
```

This request passes the quoted string to the Multics command processor for execution. This request is useful in break request lists and conditional execution lists, where the .. escape cannot be used.

Example:

```
e "ioa_ ""stopped at a break """
```

THE SYNTAX OF AN EXPRESSION

An expression can be made from variable references, constants, and probe builtin functions, which may be combined using the arithmetic operators +, -, *, and / for addition, subtraction, multiplication and division. Parenthesis may be used to indicate order of evaluation. Operations of multiplication and division are performed first, the those of addition and subtraction.

THE SYNTAX OF A VARIABLE

Variables can be simple identifiers, subscripted references, structure qualified references, and locator qualified references. Subscripts may also be expressions.

Examples:

```
date-elem                          (COBOL name)
ignatz (p -> lemma - 3)
log-type of gen-record (3)
```

The block in which a variable reference is resolved is normally determined by the source pointer, but can be altered by providing a different block in brackets after the variable name. A block may be specified in the following ways:

Example:

level N	the block and frame at level N
-N	the Nth previous invocation of the current block
LINE	the block that contains LINE, in its most recent invocation.
PROCEDURE	the block named. It may be internal to the current procedure, or external.

A block specification is meaningless for a reference to a label or entry constant unless the label or entry constant is itself being used in a block specification, in which case only the relative form (-N) is meaningful, and is taken to mean the N'th previous instance of the block designated by the label or entry constant. That is "var [sub[-2]]" references the value of var in the the second previous invocation (third on the stack) of the procedure that contains the entry or label "sub".

THE SYNTAX OF A CONSTANT

The attributes of a constant are determined by the appearance of the constant. Probe recognizes arithmetic constants (fixed or floating, binary or decimal), string constants (character or bit, in any radix from 1 to 4), and pointer constants.

The maximum length of a string constant is 256 characters.

Examples:

-123	decimal fixed point
10b	binary fixed point
45.37	decimal floating point
4.73e10	decimal floating point
4.21f10	decimal fixed point
2.1-0.3i	complex decimal
12345670o	binary fixed point entered in octal
"abc"	character string
"quote""instring"	character string with embedded quote
"1010"b	binary bit string
"FAC7"b4	hexadecimal bit string
"1222"b2	quatenary bit string
256|1200	pointer
232|7413(9)	pointer with bit offset

Note that the segment number and word offset of a pointer are specified in octal, but the bit offset, if any, is specified in decimal.

PROBE BUILTINS

Many builtin functions are provided. They can be referenced as if they were external functions, but if no argument is needed, then the argument list may be omitted. The substr and unspec builtins may be used as pseudo-variables.

addr (A)	standard PL/I builtin
addrel (P, N)	standard PL/I builtin
arglist ()	ptr to arglist of current proc or null
baseptr (N)	ptr to base of segment N
length (S)	standard PL/I builtin
linkptr ()	ptr to linkage section for current proc
maxlength (S)	standard PL/I builtin

null ()	standard PL/I builtin
pointer (P, N) or (P, N, N)	standard PL/I builtin
rel (P)	standard PL/I builtin
segno (P)	segment number of ptr P
stackframeptr ()	standard PL/I builtin
substr (S, N) or (S, N, N)	standard PL/I builtin
unspec (A)	standard PL/I builtin

In the list above, A stands for any reference to storage, N stand for any expression that yields a number, P for any expression that yields a pointer value, and S for any expression that yields a string.

If there is a program variable of the same name as a probe builtin, it can be referenced by preceeding its name with the character "%". Since this character cannot be used to construct a legal name in any of the supported languages there is no possibility of a name conflict.

Examples:

For the following examples, assume that p is declared as an aligned pointer, i as a fixed binary initial (-2), and cs as a character varying (8) initial ("abcdef").

addr (i)	the address of i
v octal (i)	displays 777776
v substr (cs, 2, 3)	"displays bcd"
let substr (cs, 4, 1) = " "	sets cs to "abc ef"
v length (cs)	displays 6
value maxlength(cs)	displays 8
v baseptr (254o)	displays 254|0

MACHINE LEVEL BUILTINS

The machine registers associated with the current stackframe may be accessed as probe builtin functions. Their names must be preceeded by a percent sign. All the machine-level builtins may be used as pseudo-variables, but the results of doing so are not specified by probe. Builtins supported, and the type of data they yield, are:

a	fixed bin (35)
e	fixed bin
q	fixed bin (35)
aq	fixed bin (71)
ea	float bin (27)
eaq	float bin
pr0 - pr7	pointer

psr	pointer
x0 - x7	fixed bin (17)

THE SYNTAX OF A LINE

A LINE is used by probe to define a source statement or a location in the object segment. It can be a label, a line number, or a special probe symbol. Lines in include files may be specified by giving the file number before the line number. The compilation listing specifies the correspondance between file numbers and source files. A statement can be specified relative to another statement. A label that looks like a line number may be specified by preceeding it with a dollar sign.

Examples:

34	line number 34
2-59	line 59 in include file 2
foo(3)	subscripted label constant
label,3	third statement after one labelled by "label"
$100	statement whose label is "100"
$c,3	the statement three statements after the current one
$b	the statement containing the breakpoint that caused the current invocation of probe.

SUMMARY OF REQUESTS

request	abbrev	function
?		lists all probe requests
.		causes probe to identify itself
..		Escape to Multics command processor
after	a	set break after a statement
args		prints arguments to current procedure
before	b	set a break before a statement
call	cl	call an external procedure
continue	c	restart break
continue_to	ct	insert temporary break and continue
display	ds	display storage in selected format
execute	e	pass string to Multics command processor
goto	g	transfer to a statement
halt	h	in break text, establish a probe level
help		print information about probe
if		execute probe requests if condition

		is true
input_switch	isw	read probe requests from switch
language	lrg	set probe langauge
let	l	assign a value to a variable
modes	mode	control probe behavior
output_switch	osw	direct probe output to a switch
pause	pa	stop a program once
position	ps	set the source pointer and print source line
quit	q	return from current probe invocation
reset	r	delete breaks
source	sc	print source lines
stack	sk	trace the stack
status	st	list breakpoints
step	s	advance one statement and halt
symbol	sb	display attributes of a variable
use	u	set source pointer
value	v	display value of an*expression
where	wh	display value of probe pointers
while	wl	execute commands while condition is true

Name: probe_modes

The probe_modes command allows the user to set the mode switches used by the probe command without invoking probe. It is intended for use in start_up.ec so the user can tailor probes behavior. It accepts precisely the same arguments as the probe "modes" request. See the MPM Commands writeup for probe for the list of accepted arguments.

Usage:

```
probe_modes MODES
```

where MODES is a list of one or more attributes, separated by spaces. If none are given, then the current modes are printed.

Once probe modes are set, they remain set until reset, even across processes. They are stored in a segment in the user's home directory called Person_id.probe, where Person_id is the user's login name. This segment should not be deleted unless the user is willing to have all modes revert to the default.