Subject: MCR10084, Revise probe.info; Enhance probe's help Request

Author: Gary Dixon

Date: 2021-02-22

Multics Ticket: #215

The new **verify_info** command found many problems in probe.info: a multi-block info segment documenting the probe command and all of its subsystem requests. These have been corrected in a revision of probe.info.

While documentation for only a few requests changed significantly, resolving the issues found by verify_info required adding Arguments, Control arguments, and List of XXX requests sections to many of the requests.

With this improved formatting for the info seg data, and the improved parsing provided by the new **help_** facility, it is now useful to add controls to the **probe** command's ***help*** request which summarize the many request formats, and can selectively display information for those request formats of interest to the user.

The probe *help* request calls help_ to display descriptions of its requests. help_ supports several control arguments that can summarize documentation.

**-brief** displays the "Syntax" section, plus item names in the "Arguments", "Control arguments", and "List of …" sections.

**-control_arg ITEM_NAME** displays the full description for items in one of those three section types whose name contains the ITEM_NAME operand.

## Proposed Changes

- Install revisions to the existing probe.info segment to correct errors reported by the new verify_info command.
- Change probe's help request (in probe_info_requests_.pl1) to use the existing help_ implementation of the -brief (-bf) and -control_arg (-ca) controls.
- Document those new controls in probe.info's block describing the *help* request.
- Correct two bugs in the *help* request that prevent the user from entering names for two existing info blocks in probe.info: the block describing the status (st) request; and the block describing the CROSS-SECTION topic. CROSS-SECTION is an argument placeholder designating an array name with subscripts which select a cross-section from an array.

The two bugs are described later in this MCR. They were found while implementing code for the new -brief and -control_arg controls.

Segments involved in this change are shown in the mbuild Build Script file that follows.

```
pr pb.mb

                    pb.mb     12/22/20  1309.4 pst Tue


Description:
 A) Change probe help request to support -brief, -control_arg and
    TOPIC = CROSS-SECTION  or  TOPIC = cross-section.
 B) Install a revision of the existing probe.info segment.

Installation_directory:  >udd>m>gd>w>pb;

 Build_script:      pb.mb;

 Bound_obj:         bound_probe_                      IN: sss  UPDATE;
   source_arch:     bound_probe_.2.s.archive           UPDATE;
     source:        probe_info_requests_.pl1           REPLACE     compiler: pl1 -ot;

 Info:              probe.info                        IN: sss.info;
                      add_name:  pb.info;
```

---

## Bug: help status

The existing probe help request code special-cases three topic names: bugs, changes, and status.

An info seg named PROGRAM_NAME.bugs.info used to be the preferred place for describing bugs in the PROGRAM_NAME program. probe.bugs.info would contain bug descriptions for the probe subsystem.

Later, the convention changed to PROGRAM_NAME.status.info. [The word “bug” was deemed inappropriate for use with customers.] But programs with an existing XXX.bugs.info segment were allowed to add an XXX.status.info name to the segment, so users familiar with the original name could still find the segment.

A few years later, another naming change switched the name for such segments to PROGRAM_NAME.errors.info; and for this change, the older XXX.bugs.info and/or XXX.status.info names were not retained.

However, probe’s *help* request was never updated to reflect the latest naming convention. If the user types:

help bugs

or

help status

the help request maps those TOPIC values to the info seg files: probe.bugs.info or probe.status.info. But neither of these files exists today. Bugs for probe are recorded in probe.errors.info.

So the probe help request will be updated to adjust the old TOPIC (bugs) to the newer TOPIC: errors. And the request will then map a pair of special names (errors and changes) onto the per-program reporting files: probe.errors.info and probe.changes.info.

The PROGRAM_NAME.changes.info files announce new features and changes to the user interface for the given PROGRAM_NAME.

probe already has a probe.errors.info segment. It does not have a probe.changes.info segment; but could have such segment at a later point in time.

The mapping of the "help status" onto a probe.status.info segment introduces a problem. probe actually has a status request which has its own documentation in an info block of probe.info. Since there is no probe.status.info segment, "help status" will no longer be special-cased. It will instead display the info block describing the status request.

---

### Bug: help cross-section

The existing probe.info segment has several info blocks that refer the reader to another block that describes how array cross-sections are referenced in probe requests. Because all the language specification manuals use the hyphenated word pair to reference a selected subset of array elements as a single item (a cross-section), probe.info denotes the placeholder arguments that reference a cross-section as a CROSS-SECTION. And the info block describing that placeholder has the names: CROSS-SECTION and cross-section (and for good measure: CROSS_SECTION and cross_section, so the info block could actually be displayed in the past).

The problem stems from all those lines in other blocks of probe.info that tell the user to type:

```
help CROSS-SECTION
```

using the placeholder uppercase and hyphenated naming convention. That is the preferred way for the user to think about and refer to such placeholders.

Unfortunately, probe's request line parser was designed for parsing arithmetic expressions. Whenever it finds a hyphen in the request line, it creates a MINUS operator token. So if a user types:

```
help CROSS-SECTION
```

the help request receives three tokens:

"CROSS" "-" [the MINUS token] "SECTION".

probe already has a routine that reassembles MINUS followed by a word token (probe calls such words IDENTIFIERs). So the fix involves calling probe_get_$control_arg to check for the new -brief and -control_arg arguments; but first the code checks whether the IDENTIFIER before that possible control arg is "CROSS" or "cross", and whether the possible control arg is "-SECTION" or "-section". If so, these tokens are joined back together and passed to help_ as info block name: CROSS-SECTION or cross-section.

## Testing

The enhancement to support the help -brief and -control_arg arguments was tested by: binding the changed code into the current bound_probe_ object; and using the repaired and enhanced probe.info segment in a directory that appears early in the process's `info_segment` search paths. By invoking this revised probe object, the new help request

could then be tested against info blocks in probe.info that include many sections displayable using -brief and -control_arg ("Syntax", "Arguments", "Control arguments" and "List of …" sections).

The following line initiates the revised probe object by all of its reference names.

```
initiate pb -a -force
r 16:06 0.177 3
```

The following command shows the revised probe object being invoked.

```
pb
command_processor_ exited at offset 2101 (level 4).
```

The first test shows use of the -brief control argument on help's position request.

```
help position -brief

2020-12-21  position, ps

Syntax:
  ps {OPERANDs}

Arguments:
 OPERAND

List of position operands:
 ps        ps +N   ps line +N   ps level N    ps level -N   ps "STRING"
 ps LINE   ps -N   ps line -N   ps level +N   ps OBJECT     ps /REGEXP/
```

The next test shows use of the -control_arg argument with an operand to select which item descriptions to display.

```
help position -control_arg level

2020-12-21  position, ps

List of position operands:
ps level N
  position to the last line executed in the block at stack level N.
ps level +N
  position to the last line executed for the block N levels more
  recent than the current stack level.
ps level -N
  position to the last line executed for the block N levels less
  recent than the current stack level.
```

The next test shows using -control_arg but forgetting to supply an operand.

```
help position -ca
probe (help): Expected argument missing. Operand for -ca
```

The next tests show the adjusting of bugs to errors, and the mapping of TOPIC errors to the info seg name: probe.errors

```
help bugs

1987-09-21  probe Known errors in the current release of probe. # Associated TR's Description

106  none
(value) When asked to print the value of a fixed(17) aligned where the
lower 18 bits are on and some of the higher bits (but not the sign bit)
are on, prints nothing for the value.
. . .
```

```
help errors

1987-09-21  probe Known errors in the current release of probe. # Associated TR's Description

106  none
(value) When asked to print the value of a fixed(17) aligned where the
lower 18 bits are on and some of the higher bits (but not the sign bit)
are on, prints nothing for the value.
. . .
```

Remapping of TOPIC changes to probe.changes shows the remapping worked, but reports probe.changes.info was not found.

```
help changes

probe: Entry not found.  probe.changes
probe (help): There is no info available for "changes".
```

The next test verifies that **help status** now displays the info block describing the probe status request.

```
help status -brief

1989-09-30  status, st

Syntax:
  st {OPERANDs} {-control_arg}

Arguments:
 OPERAND

Control arguments:
 -long, -lg   -brief, -bf

List of status operands:
 status                          status OBJECT
 status {before|after} LINE,     status *
   st {be|af} LINE,              status -all, st -a
   st {b|a} LINE
```

The final test shows that -control_arg can select and display the full description for items in brief-related info sections.

**help status -ca LINE**

```
1989-09-30  status, st

List of status operands:
status {before|after} LINE,
 st {be|af} LINE,
 st {b|a} LINE
  displays the break at LINE in current procedure.  This could be a
  line number, or one of the line reference symbols.  For details,
  type:  help LINE

  If LINE is preceded by the keyword "before" (or "be" or "b"), only
  breaks before the statement is displayed.  If preceded by "after"
  (or "af" or "a"), only breaks after the statement is removed.
```

## Documentation

probe.info is too large (more than 2000 lines long) to show in this MCR, or even to show all compare_ascii output to highlight the changes.

The most significant changes are briefly mentioned in the history comment added to this info segment.

**hcom ds probe.info**

```
>user_dir_dir>Multics>GDixon>w>pb>probe.info:
  1) change(2020-12-19,GDixon):
      A) Convert all heading line dates to iso_date format.
      B) All info blocks but the first document probe requests or
         other subsystem topics.  Therefore, none of the names on
         these info blocks should be added as external names on the
         probe.info segment.  Change the :Info: divider to :[Info]: to
         inform verify_info of this intended naming scheme.  probe's
         help request will display such "internal" info blocks using
         help_'s info_name selection value.
      C) Under verify_info rules, an info block whose first section
         title is "Syntax" with heading line date before 1985 is
         treated as documenting a command.  Since probe wants all
         such blocks treated as a Request, all of the 1979 years were
         changed to 1989; 1980, 1981, 1982, 1983, and 1984  were
         changed to 1985.
      D) Many requests had Syntax lines that referenced arguments
         without providing an Arguments section to describe them.
         Instead, they were mentioned in summary fashion in the
         Function section.  For these requests, an Arguments section
         was added in that request's info block.
      E) Enhance documentation for the "call" request.
      F) Enhance documentation for "position" and "use" requests by
         including text from MPM Commands writeup of probe.
      G) Change probe “help” request to support -brief and -control_arg
         arguments, and TOPIC values containing two hyphenated words
         (e.g., CROSS-SECTION).
r 17:34 0.432 22
```

The following shows the compare_ascii output for the enhancement to the probe “call” request documentation in probe.info.

```
A496      :Info: call: cl:
A497      02/13/80 call, cl
A498
A499      Syntax: cl PROC_NAME {(arg1, arg2, argN)}
A500
A501
A502      Function:
A503       Requires an external entry name and an argument list, which can be
A504       empty.  If an argument is of the type expected by the entry to be
A505       called, then it is passed by reference.  If the called routine
A506       changes an argument passed by reference, the variables value is
A507       changed after the call.  If probe can't determine what type of
A508       argument the entry to be called requires, or if the argument isn't of
A509       the expected type, the argument is passed by value.  Expressions are
A510       always passed by value.
A511
A512       See also "value", which is used to call external functions.
A513
A514
A515      :Info: CONSTANTS: constants:
A516      09/29/79  Syntax of constants
Changed by B to:
B569      :[Info]: call: cl:
B570      2020-12-19  call, cl
B571
B572      Syntax: cl PROC_NAME
B573              cl PROC_NAME (ARG1, ARG2, ... ARGn)
B574
B575
B576      Function:
B577       Invokes an external entry point, optionally passing as arguments one
B578       or more variables from the current program, or other EXPRESSIONs.
B579
B580
B581      Arguments:
B582      PROC_NAME
B583         names the external entry point to be called.
B584      ARG1, ARG2, ARGn
B585         each is an EXPRESSION whose value is passed as an argument to the
B586         external entry point.  The simplest EXPRESSION is the name of a
B587         variable declared by the current program.  EXPRESSIONs are
B588         documented with the value request.  For information about
B589         EXPRESSIONs, type:  help value
B590
B591
B592      Notes:
B593       Only external program entry points may be called.  The call is
B594       performed by searching for the PROC_NAME external entry point and
B595       examining the entry point descriptors for its expected arguments.  If
B596       the given ARGi is of the data type request in the descriptor, then
B597       ARGi is passed by reference.  If the called routine changes an
B598       argument passed by reference, that variable's value is changed after
B599       the call.
B600
B601       If probe can't determine what type of argument the external entry
B602       point expects, or if the argument isn't of the expected type, the
B603       argument is passed by value.  EXPRESSIONs more complicated than a
B604       single variable name are always passed by value.  For information
B605       about EXPRESSIONs, type:  help value
B606
B607
```

```
B608       For information about creating temporary variables for use in call
B609       requests, type:  help declare
B610
B611
B612      Examples:
B613       To invoke an external entry point that requires no arguments:
B614
B615         call date_time
B616         12/19/20  1752.9 pst Sat
B617
B618       To invoke an entry point with an EXPRESSION:
B619
B620         call clock ("iso_date_time")
B621         2020-12-19 17:52:19 pst
B622
B623
B624       To invoke a subroutine, passing variables declared by the current
B625       program:
B626
B627         call date_time_ (my_clock, my_time_string)
B628
B629       If my_clock is declared as fixed bin(71) aligned, then it is passed
B630       by reference; otherwise its value is converted to that data type
B631       following PL1 conversion rules.  If my_time_string is declared as a
B632       non-varying character string, then it is passed by reference;
B633       otherwise its value is converted to a temporary character string, and
B634       that temporary string is passed as the argument.  If passed by
B635       reference, then the value of my_time_string is the my_clock number
B636       converted by the date_time_ subroutine to an equivalent character
B637       representation of that clock reading.
B638
B639
B640      :[Info]: CONSTANTS: constants:
B641      1979-09-29  Syntax of constants
```

The following shows the enhanced info segment block for the probe “position” request.

**help pb.info -info position -all**

```
>user_dir_dir>Multics>GDixon>w>pb>pb.info  [Info]: position   (75 lines in info)

2020-12-21  position, ps

Syntax:
  ps {OPERANDs}

Function:
 Sets the source pointer to the statement specified and prints the
 statement.  For information on repositioning without printing the
 source statement, type:  help use

Arguments:
OPERAND
  The position request accepts many different operand types.  See a
  complete list in the following section.

List of position operands:
  Items below show the position request being used with each form
  of operand.
ps
  with no operands, position resets the source pointer to its initial
  value (same as the control pointer).
```

```
ps LINE
  positions to a LINE in the current procedure.  This could be a line
  number, or one of the line reference symbols.  For details, type:
    help LINE
ps +N
  position to the statement N statements after the current one.
ps -N
  position to the statement N statements before the current one.

ps line +N
  position to the statement N lines after the current one.
ps line -N
  position to the statement N lines before the current one.
ps level N
  position to the last line executed in the block at stack level N.
ps level +N
  position to the last line executed for the block N levels more
  recent than the current stack level.
ps level -N
  position to the last line executed for the block N levels less
  recent than the current stack level.

ps OBJECT
  position to the last line executed in the most recent invocation of
  OBJECT, where OBJECT is a program reference.  For more information,
  type:  help OBJECT
ps "STRING"
  search in the source for the next statement containing the literal
  string.  STRING may be any program label, or statement element.
ps /REGEXP/
  search in the source for the next statement that matches the qedx
  regular expression REGEXP.

Notes on positioning:

 In PL/I, names (including labels) are not available to probe when the
 source pointer is outside the block in which they are declared.  This
 means that probe cannot find a label in an internal procedure or
 begin block unless that block is now the "current block".  The label
 can still be found by a search for a character string that appears in
 the labeled line (e.g., "program_label:").

 The source segment used is either the one that the program was
 originally compiled from, if it exists, or the first segment with the
 appropriate name (NAME.pl1, NAME.fortran, etc.) found using the
 "probe" search paths.  If no source segment can be found, probe is
 unable to search in source, or print source lines.

 It is only possible to set the source pointer to an executable
 statement.  Every language has certain statements that are not
 executable.  Blank lines, comments, and declarations are not
 executable in any language.  For more information, type:
    help where
    help LINE
```

The following shows the enhanced info segment block for the probe “use” request.

**help pb.info -info use -all**

```
>user_dir_dir>Multics>GDixon>w>pb>pb.info  [Info]: use   (77 lines in info)

2020-12-21  use

Syntax:
  use {OPERANDs}

Function:
 Sets the source pointer to the statement specified and but does
 not print the statement.  For information on printing the
 source statement when repositioning, type:  help position

Arguments:
OPERAND
  The use request accepts many different operand types.  See a
  complete list in the following section.

List of use operands:
  Items below show the use request being used with each form
  of operand.
use
  with no operands, use resets the source pointer to its initial
  value (same as the control pointer).
use LINE
  positions to a LINE in the current procedure.  This could be a line
  number, or one of the line reference symbols.  For details, type:
    help LINE
use +N
  position to the statement N statements after the current one.
use -N
  position to the statement N statements before the current one.

use line +N
  position to the statement N lines after the current one.  This
  differs from the "use +N" operand when the current line contains
  more than one statement.
use line -N
  position to the statement N lines before the current one.
use level N
  position to the last line executed in the block at stack level N.
use level +N
  position to the last line executed for the block N levels more
  recent than the current stack level.
use level -N
  position to the last line executed for the block N levels less
  recent than the current stack level.

use OBJECT
  position to the last line executed in the most recent invocation of
  OBJECT, where OBJECT is a program reference.  For more information,
  type:  help OBJECT
use "STRING"
  search in the source for the next statement containing the literal
  string.  STRING may be any program label, or statement element.
use /REGEXP/
  search in the source for the next statement that matches the qedx
  regular expression REGEXP.

Notes on positioning:

 In PL/I, names (including labels) are not available to probe when the
 source pointer is outside the block in which they are declared.  This
```

```
means that probe cannot find a label in an internal procedure or
begin block unless that block is now the "current block".  The label
can still be found by a search for a character string that appears in
the labeled line (e.g., "program_label:").

The source segment used is either the one that the program was
originally compiled from, if it exists, or the first segment with the
appropriate name (NAME.pl1, NAME.fortran, etc.) found using the
"probe" search paths.  If no source segment can be found, probe is
unable to search in source, or print source lines.

It is only possible to set the source pointer to an executable
statement.  Every language has certain statements that are not
executable.  Blank lines, comments, and declarations are not
executable in any language.  For more information, type:
   help where
   help LINE
```

Finally, the revised documentation for the probe “help” request describes the new -brief and -control_arg controls.

**help pb.info -info help -all**

```
>user_dir_dir>Multics>GDixon>w>pb>pb.info  [Info]: help   (33 lines in info)

1989-09-29  help

Syntax:  help
         help *
         help TOPIC {-control_args}

Function:
 The help request prints information about probe.

Arguments:
TOPIC
  prints information about TOPIC.  Topic may be the name of a probe
  request, or the name of some other probe topic.  If TOPIC is
  omitted, help prints general information about probe.
*
  If the argument is an asterisk, help prints a list of all
  probe-related topics for which info exists.

Control arguments:
   One of the following control arguments may be given in each
   request.
-brief, -bf
   displays a summary of request syntax, arguments, and control
   arguments.
-control_arg ARG_NAME, -ca ARG_NAME
   displays full information about the given argument or control
   argument or item in a "List of ..." section.

Examples:

   help help          prints this info.
   help status        prints information about the "status" request.
   help errors        prints information about known errors in probe.
```

## Version History

| Date | Revision | Author | Comment |
|---|---|---|---|
| 2020-12-22 | 1.0 | Gary Dixon | Initial version of this MCR. |
| 2021-02-22 | 1.1 | Gary Dixon | Add missing paragraph in discussion of the Bug: help status |