MULTICS CONDENSED GUIDE

by

J. L. Bash, J. E. Hart, and M. L. Goudy

Revision 2
June 1969

CAMBRIDGE INFORMATION SYSTEMS LABORATORY
HOW TO USE THIS BOOK

The Multics Condensed Guide is intended as a quick reference book for a programmer at the console. The guide covers commands available in Limited Initial Multics (LIM).

Where possible within a given section, material is arranged alphabetically and there is only one topic on a page (a given command or request). To locate a command, simply turn to the appropriate section and look it up alphabetically.

The MCG is looseleaf; changes and additions will be made to it as appropriate. In addition, the programmer may divide the book into sections keeping only those that he needs for his console work.
How to Use This Book

I SYSTEM CONVENTIONS
   Dialing In, Logging In, and Logging Out
   System Responses
   Standard Console Activities

II INPUT STREAM
   Universal Character Conventions
   37KSR Teletypes
   1050 and 2741 Consoles

III COMMANDS
   Command Conventions
   Argument Formats for Commands
   Symbols Used in Commands

   addname  expand  print  print_dbrs
   adjust  extract_archive  print_link_info
   archive  fileio  probe
   bcpl  files
   bind  flush
   branch  fortran  qed
   change_wdir  fs_chname  read7
   chasepath  fs_readacl  remove
   contents  iocalcall  remove_dir
   convert_object  link  rename
   copy  list  runcom
   ctss_aarchv  listacl  setacl
   delacl  login  start
   delname  logout  status
   dprint  merge_edit  tape_in
   dpunch  movebranch  tape_out
   echo  mspeek  time
   edm  new_proc  unlink
   epl  nothing  who
   eplibsa
MCG TABLE OF CONTENTS

V
MERGE_EDIT CONTROL SEGMENT LINES

bcpl  error  notape
core  fetch  pure
deck  insert  symbol
entry  libe  text+link
epl  load  tmgl
eplbsa  maketl

VI
EDM REQUESTS
- backup  n next
b bottom  p print
c change  q quit
d delete  r retype
f find  s save
i insert  t top
k kill  v verbose
l locate

VII
QED REQUESTS
: absolute line  l list
a append  m move
b buffer  p print
c change  q quit
= current line  r read
d delete  k sort
e enter  x status
v exclude  s substitute
g global  y transform
i insert  w write

VIII
PROBE REQUESTS
arglist  segdump
dump_process  seginfo
info  set
initiate  stack
output  state
quit  terminate

iii Rev. 2  06019
SYSTEM CONVENTIONS

Dialing In
Logging In
Logging Out
Standard System Responses

Console Activities
  Working Directory
  Writing Source Programs
  Compilation and Assembly
  Execution
  Debugging
  Access Control
  File Transfer CTSS/Multics

Console Interrupts
SYSTEM CONVENTIONS

DIALING IN
No standard digit for dialing into the Multics system has yet been assigned. When it is, the user will follow the procedure:

- Dial appropriate digit
- Listen for high-pitched tone
- Push DATA button on 2741 and 1050.
  (Automatic on TTY 37.)
- Watch for message from Multics system to be printed out

LOGGING IN
Once Multics has responded to dialing in, the user issues a login command giving his personal name and project id as in:

```
login Smith Multics
```

The system asks for a user password. The user types in his password, for example:

```
cornco
```

Multics verifies the password and responds with R(eady) message. The user may now issue commands to Multics.

LOGGING OUT
When a user is ready to terminate a console session, he issues a logout command, i.e.,

```
logout
```

Multics responds with a W(ait), then issues the following message:

```
personal_name project_id logged out
```

as in:

```
Smith Multics logged out
```

I-2 Rev 2 06019
STANDARD SYSTEM RESPONSES

When the user dials into Multics, standard system response is:

MULTICS in operation on **date** at **time**

For example:

MULTICS in operation on Wed 8 Nov 1968 at 09:23:18.83666 EST

Multics may then follow with one or more timely messages to users. Multics terminates response with a R(eady) message giving timing information. For example:

```
  r  0.1  0.0  0
```

where:

- The first group of digits give the amount of elapsed real time.
- The second group of digits give the elapsed CPU time (used in timing commands).
- The third group of digits give the number of times a command waited for a page from a storage device.

In dialing in, as shown in the example, only the first group of digits are significant.

When the user issues a command to Multics the usual system response is a W(ait) while Multics takes appropriate action. When action is complete, Multics then issues a R(eady), for example:

```
  w  929:19.4
  r  7.6  6.2  40
```

The number following a wait gives the time of day to the tenth of a second.
CONSOLE ACTIVITIES

When a user logs into Multics, a working directory is set up for him. The directory has the form:

>user_dir_dir>personal_name.project_id

Once the working directory has been set up, the user can add entry names of segments to his directory using the branch command, or establish an entry by writing a segment in one of the editors, edm or qed.

Source programs are written in one of the Multics languages by entering either the edm or qed editor (edm or qed command) and either giving a new entry name by which the program is to be called or using an existing entry name for it. The source program segment must be named with a second component that is the name of the compiler or assembler to be used, e.g.,

  edm joe.epl
  qed sourceprog.eplbsa
  edm test.fortran

Once in an editing system, the user writes his source program in the appropriate language.

The source language program can be compiled or assembled by issuing the appropriate language command (epl, eplbsa, bcpl, tmg1, fortran), e.g.,

  epl joe
  eplbsa sourceprog
  fortran test

After assembly, a program may be executed as a command by typing its name and arguments (provided all arguments are character strings), e.g.,

  joe joedata
SYSTEM CONVENTIONS

CONSOLE ACTIVITIES

If the text segment fails to execute, the probe command and appropriate probe requests can be used to help debug the program.

Access control commands (setacl, delacl and listacl) allow the user to give other users access to contents of his working directory for reading, writing, executing, and appending to one or more of his segments.

Means have been provided in Multics to transfer segments back and forth to the older GECOS operating system and to transfer files on the 7094 CTSS system to the Multics system and vice versa. (See tape_in and tape_out commands and the merge_edit command with merge_edit control lines.)
CONSOLE INTERRUPTS (QUIT)

To stop a process or to return to command level, the user pushes the ATTENTION button once on the IBM 2741 and IBM 1050 or the INTERRUPT button once on the TTY 37.

All prior work is saved, so that the user may either enter a new system or reenter the system he was in when he issued the quit.

After pushing the appropriate button, the system prints out

```
quit
```

and then

```
(ready) and the time.
```

Example:
The user may be in EDM and wish to quit; the response may appear as follows:

```
quit
 r 1:01.1 15.1 44
```

After a quit, the user may wish to issue either a start command or a new_proc command. (The start and new_proc commands are appropriate only immediately after a quit.)

The start command allows the user to resume at the point he quit and in the system he was in when he quit, EDM in the example.

The new_proc command leaves the user in his previous working directory but creates a new process for the user. The old process is available for debugging.
Character Escape Conventions
Multics Universal Escape Conventions
   Erase and Kill
   Octal Codes
   Stylistic Convention
37KSR Teletypes
IBM 1050 and 2741 Consoles
In Multics, all characters to and from external devices are translated to ASCII by a table driven code conversion. Universal character escape conventions are provided for each type of console or card device attached to the system. However, each device may be used with stylized characters that represent some internal ASCII characters or with escape conventions unique to the device. The following pages present the Multics universal escape conventions, the stylizations, and the escape conventions used with each device.
INPUT STREAM
MULTICS UNIVERSAL ESCAPE CONVENTIONS

A. ERASE AND KILL CHARACTERS

The standard erase and kill characters are:

<table>
<thead>
<tr>
<th>Character</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>erase the previous character</td>
</tr>
<tr>
<td>@</td>
<td>delete the current line</td>
</tr>
</tbody>
</table>

B. OCTAL CODES

To represent octal codes, type a "\" (left slant) and up to three octal digits. Example:

\777

C. STYLISTIC CONVENTION

One stylistic convention holds at all consoles. The solid vertical bar (|) and the broken vertical bar (牋) are considered alternatives of the graphic for ASCII code value 174.
37KSR TELETYPES

There are no further escape conventions required for the use of the TTY37, since it uses the revised ASCII character set.

IBM 1050 AND 2741 CONSOLES

Each type ball used would require a different set of escape conventions. The ball presently implemented is the 963 type ball.

The non-ASCII characters on the 963 type ball are considered stylized versions of ASCII characters:

- @ (cent sign) for \ (left slant)
- ' (apostrophe) for ′ (accent acute)
- ~ (negation) for ^ (circumflex)

In addition, the following escapes are available:

- ` for \ (accent grave)
- < for [ (left square bracket)
- > for ] (right square bracket)
- ( for { (left brace)
- ) for } (right brace)
- ~ for ~ (overline/tilde)
## COMMANDS

### Argument Formats for Commands

### Symbols Used in Commands

<table>
<thead>
<tr>
<th>Commands</th>
<th>Symbols Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>addname</td>
<td>link</td>
</tr>
<tr>
<td>adjust</td>
<td>list</td>
</tr>
<tr>
<td>archive</td>
<td>listacl</td>
</tr>
<tr>
<td>bcpl</td>
<td>login</td>
</tr>
<tr>
<td>bind</td>
<td>logout</td>
</tr>
<tr>
<td>branch</td>
<td>merge_edit</td>
</tr>
<tr>
<td>change_wdir</td>
<td>movebranch</td>
</tr>
<tr>
<td>chasепath</td>
<td>mspeek</td>
</tr>
<tr>
<td>contents</td>
<td>new_proc</td>
</tr>
<tr>
<td>convert_object</td>
<td>nothing</td>
</tr>
<tr>
<td>copy</td>
<td>print</td>
</tr>
<tr>
<td>ctss_aarchv</td>
<td>print_dbrs</td>
</tr>
<tr>
<td>delacl</td>
<td>print_link_info</td>
</tr>
<tr>
<td>delname</td>
<td>probe</td>
</tr>
<tr>
<td>dprint</td>
<td>qdump7</td>
</tr>
<tr>
<td>dpunch</td>
<td>qed</td>
</tr>
<tr>
<td>echo</td>
<td>read7</td>
</tr>
<tr>
<td>edm</td>
<td>remove</td>
</tr>
<tr>
<td>epl</td>
<td>remove_dir</td>
</tr>
<tr>
<td>eplbsa</td>
<td>rename</td>
</tr>
<tr>
<td>expand</td>
<td>runcom</td>
</tr>
<tr>
<td>extract_archive</td>
<td>setacl</td>
</tr>
<tr>
<td>fileio</td>
<td>start</td>
</tr>
<tr>
<td>files</td>
<td>status</td>
</tr>
<tr>
<td>flush</td>
<td>tape_in</td>
</tr>
<tr>
<td>fortran</td>
<td>tape_out</td>
</tr>
<tr>
<td>fs_chname</td>
<td>time</td>
</tr>
<tr>
<td>fs_readacl</td>
<td>unlink</td>
</tr>
<tr>
<td>iocall</td>
<td>who</td>
</tr>
</tbody>
</table>

III-1 Rev 2 06019
COMMAND CONVENTIONS

ARGUMENT FORMATS FOR COMMANDS

entry represents a unique entry name (branch or link) in the user's working directory, e.g.,

my_seg

path is a general term for an argument that may represent one of the following:

(1) An entry name (branch or link) in the user's working directory or in another directory. If the entry name is in another directory, path must include enough of the pathname to the entry so that it can be found, e.g.,

my_seg (implicit path to working directory entry)

>joes_dir>zap.ep1 (explicit path to an entry in another directory)

(2) A directory, indicated by a terminating >; path must include enough of the pathname so that the directory can be found, e.g.,

>freds_dir>

where freds_dir is a unique directory in the file system.

acname is an access control name representing the name of a user or set of users. It differs from an entry name only in that it must have 3 components, personal_name, project_id, and instance_tag; for example:

Andrews.Multics.* (Tag is usually given as.*)
COMMAND CONVENTIONS

SYMBOLS USED IN COMMANDS

* Used to match any one component of a name
  (pathname, entry name, access control name)
  found in a list of names, for example,
  
  list my_dir>my_entry>*.my_seg

  might cause the listing the following
  names from the directory given as my_entry:
  
  Branches
  
  version1.my_seg
  version2.my_seg
  version3.my_seg
  newvers.my_seg
  last_try.my_seg

  ** Used in a terminating position to match
  any number of components of an argument, i.e.,
  
  list my_dir>alpha.**

  causes listing pathnames from my_dir.
  These might be:
  
  Branches
  
  alpha
  alpha.link
  alpha.beta.ep1

  Links
  
  alpha.rev >joe_dir>beta

  **
COMMAND CONVENTIONS

Separates components of entry names. Components may be without special significance, e.g.,

my_seg.a or my_seg.b

However, many components have special meanings. When writing a source program in EDM or QED, the second component of the name indicates the compiler or assembler to be used to translate the source program, e.g.,

edm my_seg.bcpl

When the command to compile the source program is given the command is:

cmpl my_seg

Compilation and assembly that follow the cmpl command create the following segments:

my_seg (text segment)
my_seg.link (linkage segment)
my_seg.symbol (symbol table)

With certain options in effect, the same command could also create:

my_seg.list (ascii listing)
my_seg.error (error segment)

When my_seg is executed, the command is simply the text segment name followed by appropriate string arguments.

The user can combine text, linkage, and symbol segments into a single object segment, e.g.,

my_seg.object

Compilation, assembly, and execution of a segment can be done with merge_edit and the GECOS system. A segment containing merge_edit instructions is set up using EDM or QED. The first component of the entry name can optionally be that of the text segment; the second component is gecos, e.g.,

edm my_seg.gecos
COMMAND CONVENTIONS

SYMBOLS USED IN COMMANDS

= Used only in the second argument of a command when similar components appear in the arguments.

= means duplicate the corresponding component of the first argument, e.g.,

    rename my_seg.ep1 =.pl

== Used only in the second argument of a command.

== means duplicate all components following as taken from the first argument, e.g.,

    rename my_seg.ep1.link your_seg.==

> Used in describing a pathname as follows:

> a an initial > designates an absolute pathname, i.e., one fixed with respect to the root directory. (The > is the abbreviated name of the root directory.)

a> a terminal > indicates the entry immediately preceding is a directory.

a>b infix > is used to show the path down to the required entry. The terminating entry may be itself a directory but as used here is treated as a terminating entry.

Note that if the path, a>b, does not begin with >, then a is presumed to be in the current working directory.

< Used in a pathname to describe motion up the directory hierarchy. > a>b<c<d means to follow the file system hierarchy down to c, then return to the directory containing c and progress down to d.

The effective result is > a>b<d. This is especially useful in the case:

<a>b

which indicates entry b in directory a in the same directory that contains the current working directory.
COMMAND CONVENTIONS

SYMBOLS USED IN COMMANDS

() Parentheses delimit a set of iteration elements. Each element in the set is inserted in turn into the enclosing command and the command is evaluated. See the following paragraph on brackets for an example.

[] Used as command delimiters. The enclosed character string is evaluated as a command and its value inserted in the command line; for example:

    print ([files *.epl])

    The files command is evaluated and the values returned inserted in the command line. When files is evaluated, the print command might read:

    print (a.epl b.epl c.epl)

    Each of the three segments will then be printed in turn by the print command (because of the parentheses permitting iteration.)

    When a command in brackets is evaluated, the value is reexamined for further delimiters.

1[] When a single vertical bar precedes a command in brackets, the value of the command is inserted into the command line but the command is not reexamined for further delimiters.
COMMAND CONVENTIONS

SYMBOLS USED IN COMMANDS

\[\] When double vertical bars precede a command in brackets, the command is evaluated but the value is not inserted into the command line. (The double vertical bar convention is the equivalent of an interjected command in the previous command language.)

Space Command elements must be delimited by spaces. Defined delimiters need not be separated from enclosed elements by spaces, e.g., \[ x \] = [x]. Absence of a space between a delimiter and the rest of an element outside the delimiter indicates concatenation, e.g., >a>b>(c d e) indicates three pathnames:

\[ a>b>c, a>b>d, \text{and } a>b>e. \]

'' Left and right accents denote a literal string.

" " Double quotation marks also denote a literal string. However, a literal string enclosed in double quotation marks cannot be nested in another literal string also enclosed in double quotation marks.

NL The end of a command is delimited by an ASCII new line character. On the TTY37 NL is indicated by pressing the LINE SPACE key; on the 2741 NL is indicated by pressing the RETURN key.

; A command delimiter, permitting commands to be stacked before execution. The commands are executed when an NL is encountered.
ADDNAME

Format: addname path entry

Purpose: To add an alternate entry name, entry to the existing entry name specified by path.

Notes: Execute and write attributes must be on in the directory containing path. Equals convention permitted in entry. entry must be unique in the directory.

Example: addname >sys_lib>Smith.Multics.epl Jones.==
where the name, Jones.Multics.epl is added to the entry, Smith.Multics.epl in the directory, sys_lib.
Adjust

Format:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>adjust path</td>
<td>Computes bit count to and truncates the segment at the last significant word. (Word containing characters other than NUL or ETX).</td>
</tr>
<tr>
<td>adjust$test path</td>
<td>Causes only printing of diagnostics that would apply if adjust were invoked for a bit count.</td>
</tr>
<tr>
<td>adjust$block path</td>
<td>Uses the current length in 1024-word blocks to calculate the segment's initial position.</td>
</tr>
<tr>
<td>adjust$block_test path</td>
<td>Causes only printing of diagnostics that would apply if adjust$block were invoked.</td>
</tr>
</tbody>
</table>

Purpose: To correct the bit count for path, a segment moved from CTSS to Multics.

(When a segment is moved, the Multics bootload or tape_daemon accepts the ETX that marks the end of CTSS file or any trailing ascii NUL characters used in padding as part of the initial bit count. The adjust command corrects this.)

Example:

```
adjust >sys_lib>ctsfil
```

where segment, ctsfil, is truncated at the last word containing a significant character. A bit count is provided.
COMANDS

Format:  archive key path entry1 ... entryn
Purpose:  To create, replace, delete, print headers of, move, or combine segments of archive segment path, where the segments are given by entry1... entryn. The name of an archive segment will have .archive appended if not already present.

key is one of the following:
  d  delete entry1...entryn from path
  r  replace old entry1 to entryn with entry1... entryn. If an entry does not exist, it is added to the end of path. An r key can be used to create an archive segment path, if none exists.
  rd remove entry1...entryn from current working directory and place them in the archive segment given by path.
  t  print headers of entry1...entryn in path. If no entries are given, all headers in path are printed.
  x  extract entry1...entryn from path and copy the entries into the directory of path. path is unchanged.

Notes:  Error messages are printed for:
  key other than d, r rd, t, or x
  path not an archive file
  path does not exist (with d or x)
  entry cannot be found or cannot be moved (e.g., entry already exists on an x request)

Secondary name .object on a segment means that text, link, and symbol segments are to be treated.

Examples:  archive r my a b cc
creates segment my.archive with components a, b, and cc.
archive d my alpha.object
deletes alpha, alpha.link, and alpha.symbol from my.archive.


**COMMANDS**

**BCPL**

Reference: BX.7.06

Format: bcpl path options

Purpose: To compile the source file, path, using BCPL.

Options:

- **old** compiler accepts old (CTSS) BCPL syntax.
- **listty** list the source segment on-line rather than in a special list segment.
- **errtty** produce source code error comments on-line rather than in the path.error segment.
- **pname** produce a cross-referenced list of occurrences of each identifier in the program as part of the source segment.
- **nobsa** BCPL does not call EPLBSA assembler when compilation is done. Text and link segments are not produced. Primary output is a compiled segment called path.eplbsa in the working directory.
- **savebsa** the compiled segment, path.eplbsa, is left in the working directory and can be assembled at a later time. Used when EPLBSA is called after compilation (no nobsa option).

Example: bcpl >system_library_1>shortprog errtty nobsa

III-11 Rev 2.06019
COMMANDS

BIND

Format: bind path

Purpose: To bind together the object segments contained in the archive segment specified by path. Two entries are created in the user's working directory by the command:

path - a bound segment composed of all the segments from path.archive.

path.map - map of the bound segment.

Note: The components of the archive segment must all be in object format and must have linkage and symbol sections.

Example: Assume alpha.archive is an archive file in the current working directory. Then

bind alpha

creates alpha and alpha.map, where alpha is the bound segment.
**BRANCH**

**Format:** branch path

**Purpose:** To create an entry name in some directory. The entry name can be specified as either a directory name (terminated by >) or an non-directory name (no terminating >). In either case the entry is designated as a branch.

**Notes:** Append mode is necessary in the directory to which path is to be added.

See LINK command for creation of links.

**Example:**

branch >user_dir_dir>dir1>dir2>

where dir2 is the directory entry added to the directory path given as

>user_dir_dir>dir1
COMMANDS

CHANGE_WDIR Reference: BX.8.14A

Format: change_wdir path
Purpose: To change the name of the user's working directory to the pathname given by path.
Examples: change_wdir >user_dir_dir>Stone.Multics
change_wdir <Martin.Multics
CHASEPATH

Reference: BX.8.13

Format: chasepath path

Purpose: To retrieve the full and final pathname of the entry, path.

Notes: Read mode required in path.

Examples: chasepath fred

If fred is an entry in a directory branch of the working directory, 
$user_dir_dir>user$, the example returns the character string:

$user_dir_dir>user>fred

If fred is a link to 
$user_dir_dir>other_user>fritz

which is a link to

$user_dir_dir>third_user>derf

then chasepath returns the character string value:

$user_dir_dir>third_user>derf

To obtain the value at the terminal give the command

echo [chasepath fred]
CONTENTS

Format: contents path
        c path

Purpose: To return as a character string the
         entire contents of the segment given
         by path.

Notes: The command makes possible the execu-
       tion of a set of commands that have been
       typed into a segment, or the selection
       from a segment of a list of arguments
       to a command, etc.

       If a segment is to be used as a set of
       commands, then each command in the seg-
       ment must be separated from the next by
       a ; (semicolon).

       There are two methods of delimiting the
       commands which are to be executed. The
       first character and last character in
       the segment path can be [ and ] respec-
       tively, in which case typing:

       contents path

       is sufficient. In the second method, the user may type:

       [contents path]

       The segment path need not begin and end
       with square brackets if the second method
       is used.

Example: Assume the following contents of seg-
         ment x:

         a b c d

Then the command:

remove ([contents x])(() .(link symbol))

removes segments:

<table>
<thead>
<tr>
<th>a</th>
<th>a.link</th>
<th>a.symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>b.link</td>
<td>b.symbol</td>
</tr>
<tr>
<td>c</td>
<td>c.link</td>
<td>c.symbol</td>
</tr>
<tr>
<td>d</td>
<td>d.link</td>
<td>d.symbol</td>
</tr>
</tbody>
</table>

III-16  Rev 2  06019
CONVERT_OBJECT

Format: convert_object path

Purpose: To convert path, path.link, and path.symbol to a single "object" segment.

Notes: If a text segment exists, an object segment is created whether or not link and symbol segments are present. Either symbol only, or link and symbol may be missing. The command comments about missing segments.

Example: convert_object alpha

If alpha was compiled in epl with alpha, alpha.link, and alpha.symbol being created at that time, then the command will create a single segment, alpha.object.
COPY

Format: copy path1 path2

Purpose: To copy the branch entry, given by path1 into the branch entry named in path2, thus creating the new branch. The entry pointed to by path2 must not exist before issuing the command.

Notes: Read mode is required for path1; write and append modes required for the directory containing path2. The equals convention may be used.

Example: copy >old_dir>fred.link george.=

Branch, fred.link, in the directory, >old_dir, is copied into the working directory as george.link.
CTSS_AARCHV

Reference: MCB-275

Format:  
ctss_aarchv path>segnam1.segnam2

Purpose:  
To extract all segments from a CTSS ascii archive file and place them in the current working directory.

segnam1 and segnam2 are the first and second components of the CTSS file name. If path is not given, the CTSS archive file containing the segment is presumed to be in the working directory.

Notes:  
ctss_aarchv finds the real names of the segments if the archive file is an epl or eplbsa file and renames the extracted segments to their real names, or if the real names are not found, they are renamed to their CTSS name.

To use the command for non-epl/eplbsa file, rename segname2 to epl or eplbsa. The resulting segments will have the following name format:

crss_name1.epl or
crss_name1.eplbsa

Example:  
ctss_aarchv alpha.eplbsa
DELACL

Format: delacl path acname1 ... acname

Purpose: To delete the access control names acname1 ... acnamen, from the ACL of the entry defined by path or from the CACL of dir if path is terminated by dir>.

path may be a path name, entry name or set of entry names defined by the * convention. acname1 ... acnamen each have the form:

personal_name.project_name.tag

where tag is the instance tag of the process-group in which user is currently working and is usually indicated by * for any instance tag. An acname might be: Brown.Multics.*

Notes: Omission of an acname argument causes an error comment.

If an acname does not appear in the appropriate ACL, delacl prints a comment and processes the next acname.

See setacl command for required access needed to delete access control names.

Example: delacl g_bcpl Doe.Multics.* Smith.Multics.*

where g_bcpl is an entry in the user's directory. Access control names Doe.Multics.* and Smith.Multics.* will be removed from the ACL of g_bcpl.

   delacl dir_bcpl> Brown.Multics.*

dir_bcpl is a directory within the user's working directory. Access control name Brown.Multics.* is removed from the CACL of dir_bcpl.

Restriction: Because delacl has not yet been implemented as described above, use the following format for deleting access control names from the CACL of a directory, some_dir.

change_wdir >some_dir
delacl "" acnames

Reference: BX.8.02

Revision: 2 06019
**COMANDS**

**DELNAME**

Reference: BX.8.10

**Format:**

delname path

**Purpose:**

To delete an entry name specified by *path*. The entry specified by *path* must not be the only name on the entry.

**Example:**

delname my_seg.list

where *my_seg.list* is an entry name in the user's working directory.
**DPRINT**

Reference: BX.5.03

Format: `dprint path1 path2 ... pathn`

Purpose: To queue segments given by `path1` to `pathn` for delayed printing by the output driver daemon.

Notes: The segments are copied onto the printer in the order given. Segments to be copied should contain Multics standard characters. If a segment is missing or has a zero length, the segment is skipped.

Example: `dprint alpha beta >joes_dir>gamma`

Segments alpha and beta from the current working directory and segment gamma from `joes_dir` directory are queued for printing by the output driver daemon.
**DPUNCH**

Reference: BX.5.03

Format:  
dpunch path1 path2 ... pathn

Purpose:  
To queue segments given by path1 to pathn for delayed punching by the output driver daemon.

Notes:  
The segments are punched in the order given. Segments to be punched should contain a binary card image in every 27 words, i.e., the 960 bits per card should reside in the first 26-2/3 words of each 27 words. If a segment is missing or has zero length, the segment is skipped.

Example:  
dpunch alpha beta >joes_dir>gamma

Segments alpha and beta from the current working directory and segment gamma from joes_dir directory are queued for punching by the output driver daemon.
**ECHO**

**Format:**
echo string

**Purpose:**
A system test command. Echo types out "echo:" and a simple character string argument or a literal string argument or the character string value of a command argument passed to echo.

**Examples:**
echo abc
echo: abc
echo a b
echo: a
echo "a b c"
echo: a b c
echo [wdir]
echo: >user_dir_dir>Shih.Multics
**COMMENTS**

**EDM**

**Format:**  edm path

**Purpose:**  To invoke the EDM editor to create or edit an ASCII segment, where:

- path is the optional pathname of a segment to be created or the required pathname of an existing segment to be edited. If only the entry name is given, the segment is assumed to be in the current working directory.

**Notes:**

- If path represents an existing segment, the EDM editor begins in edit mode; if path represents a segment to be created (or if the argument is null), the EDM editor begins in input mode.
- See the section on EDM requests for further information on input/edit modes and how to use EDM.

If the segment represented by path is a procedure for compilation or assembly, the name must include the name of the compiler or assembler to be used.

**Example:**

edm testproc.epl

The segment is in the current working directory (or will be created there). It will be an epl procedure invoked for compilation by:

\[ \text{epl testproc} \]
EPL

Format: epl entry

Purpose: To invoke the epl_daemon for EPL compilation of the source segment, entry, in the current working directory.

Notes: When the epl_daemon is invoked, the r(eady) message printed at the terminal does not indicate that compilation is complete. To determine the results of compilation as well as to check as to whether compilation is completed, the command

print epl_daemon.error

will cause printing of results of compilation. A compilation-done message will be added at a later date.

Example: epl datanal

where datanal.ep1 is the entry name of an EPL source program in the working directory.
**EPLBSA**

**Format:** eplbsa entry

**Purpose:** To invoke the epl_daemon for EPLBSA assembly of the source segment, entry, in the current working directory.

**Notes:** When the epl_daemon is invoked, the r(eady) message printed at the terminal does not indicate that assembly is complete. To determine results of assembly as well as to check as to whether assembly is completed, the command

```
print epl_daemon.error
```

will cause printing of results of assembly. (An assembly-done message will be added at a later date.)

**Example:** eplbsa my_sort

where my_sort.eplbsa is the entry name of an EPLBSA source program in the working directory.
COMMANDS

EXPAND

Reference: BX.7.05
Format: expand path mode
BY.21 • 01
Purpose: To insert into the segment, path, additional segments specified in the text of segment path.
path is scanned for statements of the form:

% include pathname

where pathname is a segment to be inserted.

Optional argument mode gives the access mode of the newly expanded path. (TREWA or any subset of the Irap, Read, Execute, Write and Append modes.)

Example: expand my_epl RW

where segment, my_epl, in the user's working directory contains the following:

my_epl: proc;
  statement1;
  % include >user_dir_dir>joe_epl;
  statement2;
  % include >user_dir_dir>make_epl;
  end my_epl;

The command causes segments, joe_epl and make_epl to be included as part of the newly expanded segment named my_epl.expanded.
**COMMENTS**

**EXTRACT_ARCHIVE**

Reference: BX.99.12

Format: `extract_archive path>seename1.seename2`

Purpose: To extract all segments from a CTSS regular archive file, and place them in the current working directory.

`seename1` and `seename2` are the first and second components of the CTSS file name. If `path` is not given, the CTSS archive file containing the file is presumed to be in the working directory.

Notes: `extract_archive` finds the real names of the segments if the archive file is an epl or eplbsa file and renames the extracted segments to their real names, or if the real names are not found, they are renamed to their CTSS name.

To use the command for non-epl/eplbsa file, rename `seename2` to `epl` or `eplbsa`. The resulting segments will have the following name format:

- `ctss_name1.epl`
- `ctss_name1.eplbsa`

Example: `extract_archive alpha.epl`
FILEIO

Format: fileio path

Purpose: To indicate that the user's next input lines are to be taken from the segment entry, specified by path, not from the console.

Notes: The format of input lines in path must be the same as if they were to be typed at the console.

When input from path is exhausted, the next input line is taken from the console.

Example: fileio >my_library>sub_loop

input will be taken from sub_loop in the directory, my_library, until the end of the segment.
FILES

Format:  files path

Purpose: To obtain a list of path names of entries within path. The command differs from the list command in that the list is returned as a character string and path names rather than entry names alone are returned.

Example: print ([files *.epl])

The files command is "nested" in the print command. When the files command is evaluated, the print command might be:

print (a.ep1 b.ep1 c.ep1)

caus[ing the three files to be printed.]
COMMANDS

FLUSH  Reference: BX.20.03

Format:  flush

Purpose: To cause all pages currently in core to be paged out. A system test command; after flushing the system, worst case timings can be obtained for command execution.


**FORTRAN**

Reference: BX.7.02

Format: fortran path

Purpose: To compile and assemble the source program segment specified by path using the FORTRAN compiler.

Example: fortran alpha

where alpha.fortran is a source program to be compiled and assembled.
FS_CHNAME

Format:    fs_chname path entry oldename newename

Purpose:    To cause one of the names of entry in the directory given by path to be replaced, deleted or added. This command interprets none of the special command symbols (e.g., *,>) and thus allows manipulation of strangely-named segments.

Notes:     When both the old entry name, oldename, and the new entry name, newename, appear in the command, newename replaces oldename.
            If oldename is the null string, "", then newename is added to the list of names for the entry.
            If newename is the null string, "", then oldename is deleted from the list of names for the entry.
            path must be a complete pathname relative to the root.

Example:   fs_chname >user_dir_dir>my_dir alpha foo ""
            One of the names of entry alpha in directory, my_dir, was foo. This entry name is deleted by the command.
FS_READACL

Format:  fs_readacl path entry

Purpose:  To cause the access control list of entry to be printed. path gives the absolute (relative to the root) pathname of the directory containing entry. The command interprets none of the special command symbols (e.g., *, >) and thus allows manipulation of strangely-named segments.

Notes:  If entry is given as a null string, i.e., "", the common access control list (CACL) of the directory given by path will be printed.

Examples:  fs_readacl >user_dir_dir>my_dir alpha
causes the ACL of segment alpha in directory, my_dir, to be printed.
fs_readacl >user_dir_dir>my_dir ""
causes the CACL of directory, my_dir, to be printed.
**COMMANDS**

**IOCALL**

Reference: BX.5.01, BF.1.01

Format: `iocall outercall ioname arguments`

Purpose: To issue I/O outer calls from command level.

- **outercall** is one of the I/O outer calls.
- **ioname** is a name used to route calls in I/O.
- **arguments** are other arguments of the given **outercall**. (Shown in table below. Hyphens show optional arguments. See BF.1.01 for argument information.)

<table>
<thead>
<tr>
<th>Outercall</th>
<th>Ioname</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>attach</td>
<td>ioname</td>
<td>type -mode- ioname2</td>
</tr>
<tr>
<td>detach</td>
<td>ioname</td>
<td>-mode- ioname2</td>
</tr>
<tr>
<td>read</td>
<td>ioname</td>
<td>worksegment -offset- -nelem-</td>
</tr>
<tr>
<td>write</td>
<td>ioname</td>
<td>worksegment -offset- -nelem-</td>
</tr>
<tr>
<td>seek</td>
<td>ioname</td>
<td>ptrname1 -ptrname2 -offset-</td>
</tr>
<tr>
<td>tell</td>
<td>ioname</td>
<td>ptrname1 -ptrname2</td>
</tr>
<tr>
<td>setsize</td>
<td>ioname</td>
<td>elementsize</td>
</tr>
<tr>
<td>getsize</td>
<td>ioname</td>
<td></td>
</tr>
</tbody>
</table>

Notes: Calls to iocall should not be programmed into procedures; use a call to outercall.

Default input and output (called user _i/o) is to a console. Either input (user_input) or output (user_output) can be diverted using iocall.

Examples: `iocall attach zz file made_up_name`

creates empty segment zz for subsequent attachment.

`iocall attach user_output syn zz`

attaches output to segment zz. A w(ait) message follows the call but no r(eady) message is given until output is reattached to console.

`iocall attach user_output syn user_i/o`

reattaches output to the console. r(eady) message follows.

`iocall detach zz`

User detaches zz after reattaching output.

III-36 Rev 2 06019
COMMANDS

REFERENCE: BX.8.04

LINK

Format: link path1 path2

Purpose: To create a link to the entry specified by path1 from the entry specified by path2.

path1 must include the directory as well as entry to enable the link to be made.

path2 may be absent. In that case a link is created in the user's working directory having the same entry name as that given in path1.

Examples:

link >library>isaac
makes a link entry "isaac" in the working directory.

link >user_dir_dir>isaac >mydir>
makes link entry "isaac" in "mydir".

link ([files >user_dir_dir>joe>**])
creates for each entry in the directory >user_dir_dir>joe a link entry of the same name in the working directory.
LIST

Format: list path ioname option

Purpose: To print out a list of entry names or a subset of the entry names in the directory given by path.

ioname is an optional name of an attached stream to which output is directed. Default is user_output. When ioname is given, diagnostics are given on both ioname_output and user_output.

If option is not present, both branches and links will be printed. The possible options are:
- b - print branches only
- l - print links only

If an option is present in the command and no ioname is to be attached, a null string, "", must be given for ioname.

Notes: * and ** conventions may be used.

Example: list system_library>alpha.**
produces a list of branches and links that might be:

Branches
- alpha
- alpha.link
- alpha.symbol

Links
- alpha.new > system_library_3_beta
LISTACL

Format:     listacl path acname1 ... acnamen

Purpose:    To print access control information on the entry name specified by path and users specified by acnames. If path terminates in >, information is printed from the directory's CACL; otherwise information is printed from the ACL of the entry name. acname has the form:

            personal_name.project_id.tag

as in: Brown.Multics.qv

Notes:     listacl "" acnames prints the CACL of the working directory relating to acnames.
listacl path "" prints access control information on all users of path.
If path specifies a directory or a branch in a directory, the user must have the read attribute on in the directory or branch. If path specifies a link, the user must have the read attribute on in the entry to which the link eventually points and must also have the execute attribute on in the directory containing the link and all intermediate directories linking to the ultimate entry.

Example:   listacl >my_dir> Smith.Multics.*

prints the CACL of my_dir for acname, Smith.Multics.*

Restriction: Because listacl has not yet been implemented as described above, use the following format to list ACL information from the CACL of a directory:

change_wdir >some_dir
listacl "" acname1 ... acnamen
**COMMANDS**

Reference: BX.3.01
not yet published

**LOGIN**

**Format:**

login **username** **project_id**

**Purpose:**
To gain access to Multics at command level after dialing into the system.

**Example:**

login Smith Multics
LOGOUT

Reference: BX.3.04

Format: logout

Purpose: To end communication with the Multics system, thus terminating a console session.

Example: logout

W 930:15.7
Smith Multics logged out

The system respond to a logout with a W(ait) followed by the time of day to the tenth of a second. The system then indicates the logout is complete; the user's connection to the computer is broken and he must dial up and login in order to restore communication.
**MERGE_EDIT**

**Format:**
merge_edit g_path runname username options

**Purpose:**
To create an IMCV tape on Multics that can be run under GECOS, performing assemblies, etc., and producing a tape by which results can be returned to Multics. (See tape_in for returning results.)

g_path specifies the entry name of a merge_edit control segment used to select text for the IMCV. (See merge-edit control lines section.) The second component of the g_path entry name, if given, must be .gecos.

runname is a 1 to 6 character primary component of the job name assigned the two tape_daemon control segments merge_edit creates in the wdir as intermediate output, i.e., a runname, jobx, creates two tape_daemon segments: jobx.control and jobx.control.binary.

username is a 1 to 12 character user name.

options Two can be specified:
- notape /* do not signal tape_daemon*/
- mh \} /* Run tape at Murray Hill or MAC*/

**Notes:**
To notify tape_daemon to execute control segments produced by a previous merge_edit the command is:
merge_edit runname (tape)

where runname is that used in the previous command and (tape) is a literal.

**Example:**
merge_edit comp2.gecos job2 Bennett mac

111-42 Rev 2 06019
COMMANDS

MOVEBRANCH

Reference: BX.8.12

Format: movebranch path1 path2

Purpose: To move a non-directory branch from one directory to another, deleting the original branch given by the entry name, path1, (and the associated ACL) and establishing the new branch with the same entry name or a new entry name, given by path2 (with the associated ACL).

Notes: Read and write modes are required in the branch to be moved. Write and execute modes are required in the directory of the branch to be moved. Write and append modes are required in the directory of the entry to be created.

If path2 already exists, no move is done.

A directory with inferior segments may not be moved.

The = convention may be used.

Examples: movebranch >old_dir>fred.link joe.=

The branch "fred.link" in directory ">old_dir" is moved to the working directory and given the name "joe.link". The entry "fred.link" in "old_dir" no longer exists.

movebranch joe.link >old_dir>==

The branch "joe.link" in the working directory, is moved to the directory "old_dir".
COMMANDS

MSPEEK

Format: mspeek path offset1 offset2

Purpose: To write onto the output stream, user_output, the octal representation of a selected part of a segment, given by path. offset1 and offset2 are character strings representing the starting and ending octal locations for the dump.

Example: mspeek my_seg 27 77
NEW_PROC

Format: new_proc

Purpose: Creates a new process and leaves the user in the working directory he was in when he logged in. The old process is available for debugging but not for further processing.

Notes: At present, new_proc causes the system to hang up the user. When he redials he will be in the new process.

Example: Assuming a quit has been made:

new_proc

creates a new process, leaving the user in his previous working directory.
NOTHING

Format: nothing

Purpose: To provide a return giving minimal return time, thus aiding the interpretation of time needed to execute other commands.

Reference BX.20.04
PRINT

Format:       print path lineno endlineno

Purpose:      To cause an ASCII text segment of entry name path, starting with the segment line specified by lineno and ending with the segment line specified by endlineno to be written in the user's output stream "user_output".

where:

path is the entry or path name of the segment to be printed.

lineno is an optional argument specifying the line number of the first line to be printed. If null, i.e., omitted or replaced with "" (2 double quotes with no space) or "" (balanced left and right accents with no space between them), in which case, the entire segment is printed, with a short identifying header. Must be used when endlineno is used. (Lineno may be a null string as above, or 0 or 1, or may be left off entirely if endlineno is also omitted.)

endlineno is an optional line number of the last line to be printed; may be omitted or replaced with "" (2 double quotes with no space) or "" (left and right accents with no space between them), in which case the segment is printed from lineno to its end.

Notes:       Assumes that new line characters are appropriately embedded in the text.

Examples:    print my_seg2 7 9
             print my_seg3

            III-47    Rev 2 06019
COMMANDS

PRINT_DBR\n
Reference BX.99.03

Format: print_dbrs

Purpose: To print out the values of the ring 0, 1, and 32 DBR settings of a single process, so that dumps of the process may be easily taken.

Notes: Currently this command is called automatically at process initialization.
**COMMANDE**

**PRINT_LINK_INFO**  
Reference: BX.9.05,BX.9.05A

**Format:**
\[ \text{print_link_info path file} \]

**Purpose:** To print linkage block information for entry `path`. Information includes:

1) text segment length.
2) `<segment>l[<symbol>]` names for each link pair.
3) list of entry point and segment definition names, giving ASCII representation, octal value and symbol class.
4) link pair list giving:
   a. address relative to linkage segment base
   b. `<segment>l[<symbol>]` to which a link points or self-reference.
   c. call pointer and argument pointer of trap word if it exists.

Presence of the optional literal file as a second argument causes the contents to be placed in a segment, `path.prlink`

**Example:**

```plaintext
print_link_info ps
Segment >user_dir_dir>Garmen.Multics ps
Text segment length (in octal)
Linkage block number 1
Entry points and segdef names
rs 32 entry point
ps 24 entry point
symbol_table 0
rel_text 30 symbol
rel_link 56 symbol
rel_symbol 64 symbol
Link pairs
10  <arg_count>l[arg_count],
12  <cv_string>l[cvc],
14  <write>l[write],
16  <read>l[read],
20  <command_arg>l[return],
22  *text10,7
40  <lib>l[lib],
```

III-49 Rev 2 06019
**PROBE**

**Format:** probe

**Purpose:** To allow the user to enter the debugging system and issue a series of requests for debugging information. The command can be issued at an interruption of a command or at normal termination of a command. Probe requests produce information on one or more segments of the process.

**Notes:** See section on probe requests for information that can be obtained.

If a request issued after a probe command is not recognizable as a request, it is treated as a command.

**Example:**

```
probe
w 924:09.8
-
```

System responds to probe command with a W(ait) followed by a hyphen. User types in the probe request immediately following the hyphen. System will print out the requested information and then issue another ready-for-request (hyphen). User terminates probe with a quit request.
**QDUMP7**

**Format:**
qdump7 path1 path2 ... pathn.

**Purpose:** To cause the segments given by path1 to pathn to be converted to 7-punch format and queued for delayed punching by the output driver daemon.

**Notes:** If a segment is missing or has a current length of zero, the segment is skipped.

**Example:**
qdump7 >user_dir_dir>Jay.Multics>comp
causes segment, comp, from the directory given in the path to be queued for punching in 7-punch format.
COMMANDS

QED

Reference: BX.9.06

Format: qed input_file output_file

Purpose: To create or to edit a text file using the QED editor.

input_file is the input stream to QED and may be an entry name in the working directory, a pathname to an entry in another directory, or can represent console input.

output_file is the output stream from QED and may be an entry name in the working directory, a pathname to an entry in another directory, or can be output to the console.

Notes: If end-of-file is reached on an input file, input switches to the console.

See section on QED requests for editing requests that can be used once QED is entered.

Examples: qed

Input and output are from and to the console.

qed my_file.bcpl

Input is taken from my_file.bcpl in the user's wdir and output is to the console.

qed >joes_dir>epl3 my_file.ep1

Input is taken from epl3 in another user's working directory and output is to my_file.ep1 in the wdir.

qed - my_eplbsa3.ep1bsa

Input is taken from the console and output is to my_eplbsa3.ep1bsa in the wdir.
read7
read7 rename

To read 7-punch card decks from a directly attached card reader into a segment with any valid pathname.

The pathname of the segment is given on the header card of the deck.

If the literal, rename, is given as an argument to read7, the user is requested to give the new pathname from the console. If a pathname given on the header card is not found, the user is requested to give a new pathname from the console.

See reference MSPM section for formatting of the input deck.

Example:
read7

If it is not possible to attach the card reader, read7 issues a comment at the console. Otherwise, read7 attaches a card reader on channel "rdrb38" and begins accepting card input.
COMMANDS

REMOVE

Format: remove path
Purpose: To remove a branch from the file system, where path terminates in the entry name of a branch.

Notes: Write mode is necessary in the branch to be deleted and its directory.
Remove refuses to remove a directory subtree (remove_dir must be used) or an entry pointed to by a link.

Examples: remove seg1
The branch seg1 is removed.

Reference: BX.8.07
REMOVE_DIR

Format: remove_dir path
Purpose: To remove the directory specified by path and all segments inferior to it.
Notes: Write mode is necessary for every branch to be removed and for the directory containing each branch.
Example: remove_dir my_dir1
The directory my_dir1 in the working directory and all segments inferior to it are removed.
RENAME

Format:    rename path entry
Purpose:   To change the entry name specified by path to the name specified by entry.
Notes:     Write attribute must be on in user's working directory. Read attribute must also be on if the * convention is used. Both * and = conventions may be used in rename.
Examples:  rename >user_dir_dir>fred george
           >user_dir_dir>george
           rename ([files *.epl]) =.pl1
           all two-component names with second component "epl" in the working directory are changed to have a second component "pl1".
RUNCOM

Format: runcom path arg1 ... argn

Purpose: To permit the user's next input lines to be taken from the ascii text segment, specified by path, rather than from the console. arg1 ... argn are optional arguments to be inserted into the text of path.

Notes: Each argument is inserted into the text of path as indicated by the include (&) sign, followed by a decimal number, where:

&1 is the first argument,
&2 is the second argument, etc.

Example: runcom >my_lib>sub_loop fred george

Assume the contents of sub_loop are:

rename >my_lib>&1 &2

Then the input from sub_loop is:

rename >my_lib>fred george

causing entry name george to replace entry name fred in the directory my_lib.
SETACL

Command Reference: BX.8.02

Format: setacl path mode acname1 ... acnamen

Purpose: To modify access to the entry name specified by path for users specified by acname1 to acnamen. The new mode is given by mode and may be any combination of letters rewa (Read, Execute, Write Append). acname has the form:

personal_name.project_id.tag

where:

personal_name is a user name, e.g., Smith,
project_id is Multics, and
tag is an instance tag identifying the process-group in which the user is working. *
(any instance tag) can be used.

Notes: setacl path "" acname1 ... acnamen
causes listed acnames to have no access to path.
setacl path mode
assumes acname is the personal_name of the invoking user and that tag is *.

If path specifies a directory or branch in a directory, the user must have read, write and execute attributes on in the directory. If the user attempts to modify an ACL, given a link to it, the user must have, besides the access above, the execute attribute on in the directory containing the link and all intermediate directories leading to the branch.

Example: setacl my_dir>alpha re Smith.Multics.*
gives read and execute access to alpha in my_dir to Smith.

Restriction: Because setacl has not yet been implemented as described above, use the following format to set access in the CACL of directory some_dir:

change_wdir >some_dir
setacl "" mode acname1 ... acnamen

III-57 Rev. 2 06019
START

Format:

start

Purpose:
To resume processing in the same process after a quit and at the point at which the quit was issued.

Example:
User is in QED building an EPL program.
User hits ATTENTION button on 2741 once.
quit
r 1:11.2 25.3 53 /*system response*/
start /*User is now in QED at the point at which he issued the quit and can resume building the EPL program.*/
COMMANDS

Format: status path

Purpose: To print detailed file status information on the branch or link specified by path.

Examples: status comp.err
branch:>user_dir_dir>1ое.Multics>comp.err
unique id: BBDHqjggWWFZMh
date used: 09/07/68 1424.4 EST Sat
date modified: 09/07/68 1351.2 EST Sat
branch modified: 09/07/68 1351.2 EST Sat
mode: read
bit length: 24192
current blocks: 1
maximum blocks: 23
ring brackets: 32 32 32
status lkx
link: >user_dir_dir>South.Multics>lkx
links to: >system_library>1ew_link
date link used: 09/07/68 1426.4 EST Sat
date link modified: 09/07/68 1426.4 EST Sat
Format: \texttt{tape\_in reel\_no segment\_list}

Purpose: To input into the user's working directory the segments, given in segment\_list, which are taken from the CTSS 7-punch format tape whose reel number is given by \texttt{reel\_no}.

segment\_list is a list of CTSS segment name pairs, written in small letters, or the list may be designated as all. If all is given, all segments on the specified tape are input.

Notes: If the segments to be input were placed on the CTSS 7-punch tape using the tape\_out command, differences in the Multics and CTSS naming conventions may have affected the names of the segments. See tape\_out command.

Examples: \texttt{tape\_in 144 all}

\texttt{tape\_in 201 epla text epla link eplb text}
Format:  `tape_out reel_no path_list`

Purpose: To write a tape containing segments from the Multics file system hierarchy onto a tape in CTSS disk editor (7-punch) format, where:

- `reel_no` is a tape identifier (name or number of the tape to be written). `scr` is used to designate any scratch tape.
- `path_list` is a list of entry names of segments to be written onto the tape. If the literal `all` is used in place of `path_list`, all segments in the working directory are written.

If `reel_no` is given without a `path_list`, a check is made to see if the control file with that identifier exists. If so, the existing control file is used to write a tape.

Notes: By CTSS/GECOS convention, all segment names have two components, each of which is 6 characters or less. A single-component name taken from Multics will be given a second component, `TEXT`. A Multics name having more than 6 characters in a given component is truncated to 6 characters, which are the first three and last three of the original name.

Example: `tape_out 25 alpha.epl beta epsilon epsilon.link`

The command produces on tape 25 four segments taken from the user's working directory and having the names:

- `ALPHA EPL`
- `BETA TEXT`
- `EPSILON TEXT`
- `EPSILON LINK`

111-61 Rev 2 06019
TIME

Format: time
Purpose: To print out five times a representation of current time in octal, followed by that time converted into ASCII representation.
Example: time
Output: 000000074423461650075004
        30 Jun 1500.47 EDT Sun 1968
        15:00:29.019652
**UNLINK**

Reference: BX.8.05

Format: `unlink path`

Purpose: To delete the link entry specified by `path`.

Notes: Write attribute must be on for the directory containing the link.

Examples: `unlink fred_link`

where `fred_link` is a link entry deleted from the user's working directory.

`unlink ([files **])`

all links in the working directory are deleted and error messages are printed for non-link entries.
Format: who username1 ...username

Purpose: To determine what users are logged into Multics.

If usernames are given as arguments, Multics will indicate whether or not those users are logged in.

Examples:

who

a list of all current users will be printed out.

who Meer Shih Spier

an indication of whether one or more of the listed users is logged in will be printed out.
MERGE_EDIT CONTROL LINES

bcpl
core
deck
entry
epl
eplbsa
error
fetch
insert
libe
load
maketl
notape
pure
symbol
text+link
tmgl
**Merge-edit Control Segment Lines**

**BCPL**

**Format:**
- `bcpl dir>sourcesegment`
- `bc dir>sourcesegment`

**Purpose:** To identify a `bcpl sourcesegment` for compilation by BCPL.

**Default:** If `dir` is null, `sourcesegment` is assumed to reside in the user's working directory.

**Example of control segment containing bcpl control line:**
- `bcpl    q_0`
- `load    q_0`
- `fetch   q_0 *`

**Merge-edit command to place bcpl program q-0 on IMOV tape:**
- `merge-edit q-0 exl shore mac`
Merge_edit Control Segment Lines

CORE

Format: core

Purpose: To obtain on-line dump of segments having either the data option or wpermt option.

Note: See pure control line for dumping of all segments.

Example: bcp1 q-0
         load q-0 wpermt
         core
         fetch q-0 *
**DECK**

**Format:**
```
deck segname1 ... segnamen
```

**Purpose:**
To obtain punched decks of object code resulting from BCPL, EPL, EPLBSA, or TMGL activities or maketl and text+link inclusions conducted on segname1 to segnamen.

**Default:**
If one of the segnamei is *, all object code generated from source segments in a given run will be punched.

**Example:**
```
bcp1 q-0
bcp1 x-0
load q-0
load x-0
fetch q-0 * x_0 *
decq q-0
```

**Commands:**
```
merge_edit q-0 ex1 stone mac notape
merge_edit n-0 ex2 stone mac
```
**Merge_edit Control Segment Lines**

**ENTRY**

**Format:** entry segmentname entryname

**Purpose:** To specify an entry point for the start of execution of the pseudo-process. Segmentname is the name of an external segment and entryname is the entry point within segmentname.

**Default:** If entryname is null, entryname is presumed to be the same as segmentname.

**Notes:** segmentname entryname in merge_edit control line is equivalent to segmentname$entryname in EPL and to <segmentname> | [entryname] in EPLBSA.

**Example:**

epl syzygy
entry syzygy sunspot
load syzygy
fetch syzygy *

IV-5  Rev 2 06019
EPL

Format:  
\texttt{epl directory>sourcesegment}
\texttt{e directory>sourcesegment}
\texttt{epl sourcesegment}
\texttt{e sourcesegment}

Purpose:  To identify a \texttt{sourcesegment} to be compiled by EPL.

Default:  If \texttt{directory} is null, \texttt{sourcesegment} is assumed to reside in the user's working directory.

Example:  
\texttt{epl syzygy}
\texttt{load syzygy}
\texttt{fetch syzygy *}


Merge_edit Control Segment Lines

EPLBSA

Format:  
eplbsa directory>soucesegment
        eb directory>soucesegment
        eplbsa soucesegment
        eb soucesegment

Purpose:  To identify a source segment to be assembled using EPLBSA.

Default:  If directory is null, source segment is assumed to be in the user's working directory.

Example:  
eplbsa quirk
         load quirk
         fetch ***
ERROR
Format:   error
Purpose:  To cause the error segment to be printed on the on-line output.
Note:     An error segment is automatically returned to the user if a fetch control line is in effect.
Example:  
  eplbsa quirk
  load quirk
  deck *
  error
**Merge edit Control Segment Lines**

**FETCH**

**Format:** fetch seg1 desc1 ... segn descn

**Purpose:** Fetches (returns) object (assembled) segments, placing entries for them in the user's working directory.

seg1 is the first component of a source segment name (CTSS name1) to be put in the working directory. If seg1 is ", all segments produced by compiler and assembler activities are fetched.

desci is any character string or *. If *, text, link, symbol, and list segments are fetched. If desci is any other string, no list segment is returned.

**Note:** If desc of the last segment is blank, text, link, and symbol segments are returned.

**Examples:**

<table>
<thead>
<tr>
<th>EXAMPLE1</th>
<th>EXAMPLE2</th>
</tr>
</thead>
<tbody>
<tr>
<td>epl syzygy</td>
<td>epl syzygy</td>
</tr>
<tr>
<td>eplbsa quirk</td>
<td>eplbsa quirk</td>
</tr>
<tr>
<td>load syzygy</td>
<td>load syzygy</td>
</tr>
<tr>
<td>load quirk</td>
<td>load quirk</td>
</tr>
<tr>
<td>fetch syzygy m quirk</td>
<td>fetch *</td>
</tr>
</tbody>
</table>

In example1, working directory entries are made for syzygy and quirk with text, link, and symbol segments returned for each. A list segment is also returned for quirk.

In example2, working directory entries are made and text, link, and symbol segments returned for syzygy and quirk.

IV-9 Rev 2 06019
**Insert**

Format: `insert dir>name_gecos`

Purpose: To insert a previously created `merge_edit` control segment, `name_gecos`, in the current control segment. When an insert line is encountered, control lines are read from `name_gecos`. When the final `name_gecos` line is read, control lines are again read from the current control segment.

Notes: If `dir>` is null, `name_gecos` is assumed to be in the working directory. Nesting of insert lines is permitted to a depth of 9.

Example: Control segment `syzygy_gecos` contains:

```
ep1 syzygy
load syzygy
fetch syzygy
```

This sequence of control lines:

```
bcpl q_0
load q_0
insert syzygy_gecos
fetch q_0
```

produces the following sequence of `merge_edit` control lines:

```
bcp1 q_0
load q_0
epl syzygy
load syzygy
fetch syzygy
fetch q_0
```

IV-10  Rev 2 06019
LIBE

Format:  `libe  se~name  options`

Purpose: To load a segment directly from a library file.

Options:

<table>
<thead>
<tr>
<th>Option</th>
<th>Meaning</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>f0</td>
<td>Directed fault 0</td>
<td>00</td>
</tr>
<tr>
<td>f1</td>
<td>Directed fault 1</td>
<td>10</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>f7</td>
<td>Directed fault 7</td>
<td>70</td>
</tr>
<tr>
<td>data</td>
<td>Data segment</td>
<td>01</td>
</tr>
<tr>
<td>slvprc</td>
<td>Slave procedure</td>
<td>02</td>
</tr>
<tr>
<td>exonly</td>
<td>Execute only</td>
<td>03</td>
</tr>
<tr>
<td>masprc</td>
<td>Master procedure</td>
<td>04</td>
</tr>
<tr>
<td>slvacc</td>
<td>Slave access</td>
<td>20</td>
</tr>
<tr>
<td>wpermt</td>
<td>Write permit</td>
<td>40</td>
</tr>
</tbody>
</table>

Segment descriptor bits are taken as the inclusive OR of option bits. A descriptor containing fault 2 is created for a segment given slvacc. Options slvacc and wpermt are used only with other options, as in:

Control Line

<table>
<thead>
<tr>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>libe prog1 slvacc, slvprc, wpermt 62</td>
</tr>
</tbody>
</table>

Initial default option values are slvacc, slvprc. If a control line is encountered having other options, previous option values are cleared. The new values become default values for subsequent lines until changed.

Notes: Segments called off library automatically, not requiring libe control lines:

`escape free_page_pool f2catc`
`get_put grow init`
`length Library_dictionary linker`
`messag newpag newseg`
`relpag search segman`
`segpr tracerdatabase trunc`

Example:

`eplbsa quirk`
`load quirk`
`li bin_oct`
`fetch * *`
LOAD

Format:  load * sourcesegment options
         ld sourcesegment options

Purpose: To cause the loading and execution of segments produced by BCPL, TMGL, EPL or EPLBSA activity.

Default: An asterisk (*) for sourcesegment causes all text, link, and symbol segments produced BCPL, TMGL and EPL compilations and EPLBSA assemblies in the same run to be loaded.

If options are null, options specified by the last load, libe, text+link, or maketl control line remain in effect.
If no options were previously specified, default options are slvprc slvacc.

Options: Options are given under libe control line.

Example: epl syzgy
         eplbsa quirk
         load *
         fetch **
Merge edit Control Segment Lines

MAKETL

Format: maketl dir>entryname:se~name options
         mk dir>entryname:se~name options

Purpose: To cause a previously assembled text segment, entryname, in directory, dir, to be loaded as segment se~name, for execution. A dummy linkage segment is also loaded.

Default: If dir is null, the user's working directory is presumed.
         If se~name is null, the text segment is loaded as entryname.
         If options are null, options specified by the last load, libe, text+link, or maketl control line remain in effect. If no options were previously specified, the default options are slvprc slvacc.
         If dir>entryname is *, a dummy text segment is loaded.

Note: Control line, text+link, loads a text segment with its actual linkage segment.

Options: Options are given under libe control line.

Example: maketl spasm
NOTAPE

Format: notape

Purpose: To suppress the creation of a return tape by GECOS, overriding a fetch control line.

Notes: This control line differs from the notape argument in the merge-edit command. The argument prohibits Multics from making an IMCV for input to GECOS: the control line causes GECOS to suppress creation of a return tape.

Example:

eepl syzygy
load syzygy
deck *
notape
Merge edit Control Segment Lines

PURE

Format: pure

Purpose: To obtain on-line dump of all segments if a core control line is also included.

See core control line for on-line dump of data and write-permit segments only.

Examples: epl syzygy

load syzygy

pure

core
SYMBOL

Format: symbol

Purpose: To cause the symbol segment of all segment groups named in text+link control lines to be loaded. If this line is absent, only text and link segments will be loaded.

Note: See text+link control line.

Example: tl spasm

symbol
Merge edit Control Segment Lines

TEXT+LINK

Format:  text+link dir>entryname:segname options
tl dir>entryname:segname options
tl dir>entryname options

Purpose:  To cause a previously assembled text segment, entryname, in directory, dir, to be loaded as segment, segname, for execution. The associated linkage segment is also loaded.

Default:  If dir is null, the user's working directory is presumed.
If segname is null, the text segment is loaded as entryname.
If options are null, options specified by the last load, libe, text+link, or maketl line remain in effect.
If no options were previously specified, default options are slvprc slvacc.
If dir>entryname is *, a dummy text segment is loaded.

Options:  Options are given under libe control line.

Note:  Control line, maketl, loads a text segment and a dummy linkage segment.

Example:  tl spasm
TMGL

Format:  `tmgl dir>sourcesegment`

Purpose: To identify a `sourcesegment` for compilation by TMGL.

Default: If `dir` is null, `sourcesegment` is assumed to be in the user's working directory.

Example: `tmgl scan
load scan
fetch scan *`
EDM Requests

BACKUP: -

Format: -n

Purpose: Move pointer back up the segment the number of lines specified by the integer n.

Spacing: A blank is optional between the request and the integer argument.

Default: If n is null, the pointer is moved up one line.

Example:

Before: 
a: procedure;
   x = y;
   q = r;
   s = t;
   -> end a;

Request: -2

After: 
a: procedure;
   x = y;
   -> q = r;
   s = t;
   end a;
EDM Requests

**BOTTOM:** b

**Format:** b

**Purpose:** Move pointer to end of segment and switch to EDM input mode.

**Pointer:** Set after last line in file.

**Example:**

Before:  
```  a: procedure;  
  x = y;  
  q = r;  
  s = t;  
  end a;  
```

Request: b

After:  
```  a: procedure;  
  x = y;  
  q = r;  
  s = t;  
  end a;  ```
EDM Requests

CHANGE: c
Format: cn/string1/string2/
Purpose: Replace string1 by string2 in the number of lines indicated by integer n. EDM responds to each change by printing the line with the changed text in red if the user is in VERBOSE mode.

Delimiters: Any character not appearing in string1 or string2 can delimit the strings (/ is shown in the format). Delimiter following string2 is optional. A space before n and between n and the string1 delimiter is optional.

Default: If integer is absent, only string1 of the current line is changed.
If string1 is absent, string2 is inserted at beginning of line.

Pointer: Set to last line changed.

Example:

Before: -> a: procedure;
         x = y.
         q = r.
         s = t;
         end a;

Request: c2././;
Response: x = y;
         q = r;

After: a: procedure;
         x = y;
         q = r;
         s = t;
         end a;

V-5 Rev 2 06019
EDM Requests

DELETE d

Format: d n

Purpose: Causes the number of lines given by the integer n to be deleted. Deletion begins at the current line.

Spacing: A space is optional between d and the integer.

Default: If n is null, the current line is deleted.

Pointer: Set to first line following the lines deleted.

Example:

Before: a: procedure;
   -> x = y;
    q = r;
    s = t;
    end a;

Request: d 2

After: a: procedure:
   -> s = t;
    end a;
FIND  
f
Format:  f string
Purpose:  Search segment for line beginning with string. Search starts at line following the current line and continues around the entire segment until string is found or until return to current line. The current line is not searched. If line is not found, an error message, NO, is printed in red. If the line is found and user is in VERBOSE mode, the line is printed.

Spacing:  A single blank following f is not significant; all other leading and embedded blanks are used in searching.

Default:  If string is null, EDM searches for the string requested by the last f or l request.

Pointer:  Set to line found or remains at current line if the line is not found.

Example:  ↓ (first character position)
Before:  a: procedure;
          -> x = y;
          s = t;
          end a;

Request:  f  t  (note blanks for character positions)
Response:  NO
Request:  f  s
Response:  s = t;  (VERBOSE mode)
After:   a: procedure;
          x = y;
          -> s = t;
          end a;

V-7  Rev 2 06019
**EDM Requests**

**INSERT:**
- i

**Format:**
- i newline

**Purpose:**
Insert **newline** after the current line.

**Spacing:**
First blank following i is not significant. All other leading and embedded blanks become part of the text of the new line.

**Default:**
If **newline** is null, blank line is inserted.

**Pointer:**
Set to the inserted line.

**Note:**
Immediately after a t (TOP) request, an i request causes the newline to be inserted at the beginning of the segment.

**Example:**

**Before:**

```plaintext
↓ (first character position)
a: procedure;
  x = y;
  q = r;
-→ s = t;
```

**Request:**
i end a;

**After:**

```plaintext
a: procedure;
  x = y;
  q = r;
  s = t;
-→ end a;
```
EDM REQUESTS

KILL: k
Format: k
Purpose: To inhibit EDM from printing out responses following an f, l, or c request. The EDM system default is VERBOSE mode.

Pointer: Unchanged.
Note: See v (VERBOSE) request.

Example:
Request: v
  c /y/z
Response: y = z;
Request: k
  c /z/y
No response
LOCATE: 1

Format: 1 string

Purpose: Search segment for line containing string. Search starts at line following current line and continues around entire segment until string is found or until return to current line. If the line is not found, an error message NO is printed out in red. If line is found and user is in VERBOSE mode, the line is printed.

Spacing: Single blank following 1 is not significant. All other leading and embedded blanks are used in searching.

Default: If string is null, EDM searches for the string requested by the last l or f request.

Pointer: Set to line found or remains at current line if line not found.

Example:
Before: a: procedure;
   x = y;
   q = r;
   s = t;
   end a;

Request: 1 x =

After: a: procedure;
   x = y;
   q = r;
   s = t;
   end a;

V-10 Rev 2 06019
EDM Requests

NEXT: n

Format: n n

Purpose: Move pointer down the segment the number of lines specified by the integer n.

Spacing: Blank optional between n and the integer.

Default: If integer n is null, the pointer is moved down one line.

Example:

Before: a: procedure;
   x = y;
   q = r;
   s = t;
   end a;

Request: n

After: a: procedure;
   x = y;
   q = r;
   s = t;
   end a;
EDM Requests

PRINT: p
Format: pn
Purpose: The number of lines specified by the integer n will be printed out beginning with the current line.
Spacing: A blank is optional between p and the integer.
Default: If n is null, the current line is printed.
Pointer: Set to last line printed.
Example:

Before: a: procedure;
       x = y;
       q = r;
       s = t;
       end a;

Request: p 3
Response: q = r;
          s = t;
          end a;
After: a: procedure;
       x = y;
       q = r;
       s = t;
       -> end a;
QUIT: q
Format: q
Purpose: Terminate EDM editing without saving the edited copy of the segment.
Note: To save segment, see s (SAVE) request.

Example:
Oldfile: a: procedure;
   x = y;
   q = r;
   s = t;
   end a;

Request: c /;/ (y,r)/;
p
Response: a: procedure (y,r);

Request: q
Newfile: a: procedure;
   x = y; \{ Original (unedited) file is retained. \}
   q = r;
   s = t;
   end a;
**RETYPE**: r

**Format**: r newline

**Purpose**: Replace current line with newline.

**Spacing**: One blank between r and newline is not significant. All other leading and embedded blanks become part of the text of the new line.

**Default**: If newline is null, a blank line replaces the current line.

**Pointer**: Unchanged.

**Example**:

**Before**:
```
  a: procedure;
  -> x = y;
  q = r;
  s = t;
  end a;
```

**Request**: r dcl (r,t) float bin (27);

**After**:  
```
  a: procedure;
  -> dcl (r,t) float bin (27);
  q = r;
  s = t;
  end a;
```
SAVE:  s
Format:  s  path
Purpose:  To terminate EDM editing and save the edited copy.  path can give the directory and the entry name within the directory under which the segment is to be saved.  If only the entry name for the saved copy is given, the working directory is assumed.
Spacing:  A blank between s and path is not significant.
Default:  If path is null and if the original name of the segment is not null, the edited segment is saved under the original name; the original segment is deleted.  If path is null and no previous segment exists, an error message is printed and EDM looks for another request.
Note:  To terminate editing without saving the edited copy, see q (QUIT) request.
Example:
Oldfile:  a: procedure;
         x = y;
         q = r;
         s = t;
         end a;

Requests:  c/; (y,r);/
Newfile:  a: procedure (y,r);
         x = y;
         q = r;
         s = t;
         end a;

Edited file is retained.

V-15 Rev 2 06019
EDM Requests

TOP: t
Format: t
Purpose: Moves pointer to first line of segment.
Pointer: At first line of text.
Note: An i (INSERT) request immediately following a t request causes insertion of a text line at the beginning of segment. See INSERT.

Example:
Before: a: procedure;
         x = y;
         q = r;
         s = t;
         end a;
Request: t
After:  a: procedure;
         x = y;
         q = r;
         s = t;
         end a;
EDM REQUESTS

VERBOSE: \( v \)

Format: \( v \)

Purpose: Causes EDM to print out responses following an \( f \), \( l \), or \( c \) request. The default EDM mode is VERBOSE.

Pointer: Unchanged.

Note: See \( k \) (KILL) for inhibiting VERBOSE mode.

Example:

Before:  
a: procedure;
    \( x = y; \)
    \( \rightarrow q = r; \)
    \( ss = t; \)
    end a;

Requests: \( v \)
        \( c/ss=/s=/ \)

Response: \( s = t; \)

After:  
a: procedure;
    \( x = y; \)
    \( \rightarrow q = r; \)
    \( s = t; \)
    end a;
<table>
<thead>
<tr>
<th>REQUEST MEANING</th>
<th>REQUEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>absolute line address</td>
<td>:</td>
</tr>
<tr>
<td>append</td>
<td>a</td>
</tr>
<tr>
<td>buffer</td>
<td>b</td>
</tr>
<tr>
<td>change</td>
<td>c</td>
</tr>
<tr>
<td>current line address</td>
<td>=</td>
</tr>
<tr>
<td>delete</td>
<td>d</td>
</tr>
<tr>
<td>enter</td>
<td>e</td>
</tr>
<tr>
<td>exclude</td>
<td>v</td>
</tr>
<tr>
<td>global</td>
<td>g</td>
</tr>
<tr>
<td>insert</td>
<td>i</td>
</tr>
<tr>
<td>list</td>
<td>l</td>
</tr>
<tr>
<td>move</td>
<td>m</td>
</tr>
<tr>
<td>print</td>
<td>p</td>
</tr>
<tr>
<td>quit</td>
<td>q</td>
</tr>
<tr>
<td>read</td>
<td>r</td>
</tr>
<tr>
<td>sort</td>
<td>k</td>
</tr>
<tr>
<td>status</td>
<td>x</td>
</tr>
<tr>
<td>substitute</td>
<td>s</td>
</tr>
<tr>
<td>transform</td>
<td>y</td>
</tr>
<tr>
<td>write</td>
<td>w</td>
</tr>
</tbody>
</table>
QED REQUESTS

QED EDITOR
The qed editor performs operations on text in a working space called a buffer. A buffer contains zero to any number of lines of text, and there may be any number of buffers. Each buffer is identified by a name. There is one current buffer; all other buffers are auxiliary buffers.

BUFFER NAMES
The buffer name can be any length but only the last five characters are significant. Generally, buffers are named with a one to five character name enclosed in parentheses. If the name is one character long, and not a carriage return or apostrophe, the parentheses can be omitted (e.g., buffer names X and (X) are identical.)

TEXT ADDRESSING
QED accepts commands and text as a stream of characters from the console. Text within the current buffer is specified by (1) line addresses or (2) strings (regular-expressions) in the text line.

Lines in the current buffer may be addressed in the following ways:
1. by current line number
A decimal number not beginning with "0" or an octal number beginning with "0" is interpreted as a current (relative) line number. The first line is numbered 1, the second 2, the tenth line 10 or 012, etc. This number may change during editing. Example:

3,6 p
means print lines 3 to 6, inclusive.

VI-2 Rev 2 06019
QED REQUESTS

TEXT ADDRESSING (CONT.)

2. by absolute line number
   The character ' (apostrophe immediately
   followed by a decimal number (or octal
   number beginning with "0") is interpreted
   as an absolute line number. This number is
   assigned to each line in the current buffer
   when the text is initially read into the
   buffer from a segment. These line numbers
   never change except after read requests
   (which cause a new set of absolute line
   numbers to be assigned to text in the
   buffer). New lines created during editing
   have undefined absolute line numbers.
   The character "" not followed by a digit
   causes a search for the first undefined
   absolute line after the current line.
   (The search is cyclic from the line after
   the current line to the current line.)
   If there is no line with the given
   absolute line number an error message is
   printed on the console (see "Diagnostics").
   Example:

   '53 p

   means print the text on the line designated
   by absolute line number 53.

3. by the value of the current line (" .")
   The character " ." (period) in a QED address
   means the value of the current line. This
   value is changed by most edit requests.
   Example:

   . p

   means print the current line. In the
   examples provided for each request an
   arrow (<-) indicates the position of " ."
   (the value of the current line).

VI-3 Rev 2 06019
QED REQUESTS

TEXT ADDRESSING (CONT.)

4. by the special character "\$"
The value of \$ in an address is the last line
of text in the buffer. This value may change
during editing. Example:

\[ 1,\$ \ p \]
means print all lines from line 1 to the last
line.

5. by context
The string, \texttt{/regular expression/}, causes a
search by QED to match \texttt{regular expression} in
the text. The search begins at the line
after the current line and cycles to the
current line. If the search is successful,
the first occurrence of \texttt{regular expression}
(in the direction searched) has been located.
Example:

\[ /x=2y/ \ p \]
causes the first line of text containing
"x=2y" to be printed and causes "." (current
line pointer) to be set at that line.

6. by additive combinations of methods 1. to 5.
An address followed by + or - followed by
another address (normally relative line
number or regular expression) can be used to
address a line.

\[ 40+4 \ p \quad \text{print line 44} \]
\[ /xyz/-5 \quad \text{print a line five lines before the} \]
\[ \text{line containing the regular} \]
\[ \text{expression, xyz.} \]

VI-4 Rev 2 06019
QED REQUESTS

REGULAR EXPRESSIONS

Conventions used in writing regular expressions in QED can best be shown by examples. These are:

/a/ matches letter "a" anywhere on a line.
/abcd/ matches string "abcd" anywhere on a line.
/ab*c/ matches strings "ac", "abc", "abbc", "abbbc", ...
/abcdefghijklmnopqrstuvwxyz/ matches string "abc" or string "def".
/(ilo)nto/ matches strings "into" and "onto".

In addition, the characters "", ".", and ":" have special meaning. The character " " matches the zeroeth character on a line. The character ":" matches the character after the last character on a line. The character ":." matches any character on a line. For example:

/./ matches an entire line regardless of length.
/begin/end$/ matches a line beginning with "begin" or ending with "end".
/in.*to/ matches a line containing "in" and "to" in that order.
/beg.*end$/ matches a line starting with "beg" and ending with "end".
/\$/ matches a blank line.
/\$/ is an illegal combination matching nothing.

TEXT INPUT

A number of QED requests are followed by literal text input. This text must be preceded by a space or a carriage return. The text consists of any string of characters terminated by \f. The \f is not part of the text but delimits end of text; \f is used at the beginning of the next line following the last character in the body of the text.

ESCAPE CHARACTERS

Standard Multics escapes are used. See INPUT STREAM section.

VI-5 Rev 2 06019
QED REQUESTS

DIAGNOSTICS

01  Regular Expression search failed.
02  Unrecognized request or address.
03  Regular Expression syntax error.
04  Address syntax error.
05  Address wrap around.
06  Address out of buffer.
07  Abs line search failed.
08  File system error.
09  Request syntax error.
16  Unknown Regular Expression type.
17  Out of memory.
18  Overflow on store.
19  Passed EOF on store.
20  Free of block 0.

(Diagnostics 16 to 20 are fatal.)
QED REQUESTS

**ABSOLUTE LINE NUMBER**:  
Format:  \( \text{adri:} /\text{regular_expression}/ : \)

Purpose: Prints absolute line number of the line addressed.

"." value: set to the addressed line.

Note: If the absolute line number is undefined, a "?" is typed.

Example:

<table>
<thead>
<tr>
<th>Buffer Contents</th>
<th>Absolute line number</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x = y; )</td>
<td>'51</td>
</tr>
<tr>
<td>( \text{if } y &lt; 10 )</td>
<td>'52</td>
</tr>
<tr>
<td>( \text{GO TO PROCA;} )</td>
<td>'53</td>
</tr>
</tbody>
</table>

Requests:  
\[ /<10/d \才可以 \]
\[ /\text{PROC}/ : \]

Results:  
'53

Even though line 52 is deleted the absolute address is the same as it was when the line was created, i.e., absolute lines are not changed by editing.
QED REQUESTS

APPEND a

Format: astra or \regular_expression/a
text text
\f \f

Purpose: To append text after the line addressed.

"." value: Pointer set to last line appended. If no lines were appended, "." is
set to the line addressed.

Default: a is the same as .a

Example:

Before:

a: procedure; Line 1
x = y; Line 2
end a; Line 3 <-

Request:

2a or /y/a
q = r; q = r;
\f \f

After:

a: procedure Line 1
x = y; Line 2
q = r; Line 3 <-
end a; Line 4

VI-8 Rev 2 06019
QED REQUESTS

BUFFER
b
Format: bx
where x is the name of a buffer.
Purpose: To make the current buffer an auxiliary buffer, and to make x the current buffer. If buffer x does not exist it is created.

Initially, buffer 0 is the current buffer.

"." value: Line pointer unchanged in each buffer.

Example:
Before:
Buffer 2 is the current buffer.
"." is at line 2 in buffer 2.

Buffer 3 was previously the current buffer; at that time, "." was at line 3 in buffer 3.

Request:
b3

After:
Buffer 2 is an auxiliary buffer, and buffer 3 is the current buffer. "." in buffer 3 is at line 3.

If the request b2 is issued later, buffer 2 will be the current buffer, and "." will be at line 2 in buffer 2.
QED REQUESTS

CHANGE c

Format: adr1,adr2c
text \f

Purpose: To delete the lines specified by adr1 through adr2 and to substitute (input) other text for the deleted lines.
(The line number specified by adr1 must not exceed adr2.)

"." value: Line pointer is set to the last line of the text. If no lines of text are substituted (input), "." is set to the line before the first line deleted.

Default: adr1c is the same as adr1,adr1c
c is the same as .c

Note: adr2 must be greater than or equal to adr1 (i.e., the addressed lines cannot cross zero cyclicly.)

Example: Before:

a: procedure;
  x = y;
  q = r;
  end a;

Line 1
Line 2
Line 3
Line 4 <-

Request:

2,3c or /x/,/q/c
s = t;
  s = t;
u = v;
  u = v;
w = z;
  w = z;
\f
\f

After:

a: procedure
s = t;
u = v;
w = z;
end a;

Line 1
Line 2
Line 3
Line 4 <-
Line 5
QED REQUESTS

CURRENT LINE NUMBER =
Format: /regular expression/=  
Purpose: Prints current value of a line. 
"." value: Set to the addressed line. 
Default: = is the same as $=$  
Example:

<table>
<thead>
<tr>
<th>BUFFER CONTENTS</th>
<th>CURRENT LINE NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>A: procedure;</td>
<td>0001</td>
</tr>
<tr>
<td>declare x fixed bin(17);</td>
<td>0002</td>
</tr>
<tr>
<td>a=b;</td>
<td>0003</td>
</tr>
<tr>
<td>y=x;</td>
<td>0004</td>
</tr>
<tr>
<td>x=x+1;</td>
<td>0005</td>
</tr>
<tr>
<td>end A;</td>
<td>0006 &lt;-</td>
</tr>
</tbody>
</table>

The current line is line 6 and the request:  
= causes:  
0006  
to be printed at the console, and the value of "." remains at 0006. 
Similarly, the request:  
/y=x/= causes:  
004 to be printed at the console and the value of "." is set to 0004.
QED REQUESTS

DELETE  
d

Format:  adr1,adr2d

Purpose:  To delete the lines specified by adr1 through adr2. (The line number specified by adr1 must not exceed adr2.)

"." value:  Line pointer is set to the line after the last line deleted.

Default:  adr1d is the same as adr1,adr1d
d is the same as .d

Note:  adr2 must be greater than or equal to adr1. (i.e., the addressed lines cannot cross zero.)

Example:

Before:

a: procedure;  Line 1
 x = y;  Line 2
 q = r;  Line 3
 s = t;  Line 4
 end a;  Line 5 <-

Request:

3,4d  or  /q/,/s/d

After:

a: procedure;  Line 1
 x = y;  Line 2
 end a;  Line 3 <-

VI-12 Rev 2 06019
ENTER

Format: e/regular expression/name/

Purpose:
To tag a regular expression with a specified name. If the same name is used in several ENTER requests, the most recent request takes precedence.

Examples:

In sequence A, the SUBSTITUTE request replaces regular expression, henry with the string, aldrich. In sequence B, the SUBSTITUTE request replaces regular expression whose tag is henry, i.e., alpha, with the string, aldrich. <> symbols indicate that the string enclosed is a tag.

Sequence A
Buffer Contents: alpha = henry;
Request Sequence: e/alpha/henry/ s/henry/aldrich/
Buffer Contents: alpha = aldrich;

Sequence B
Buffer Contents: alpha = henry;
Request Sequence: e/alpha/henry/ s/<henry>/aldrich/
Current Line: aldrich = henry;
EXCLUDE

Format: adr1,adr2vrequest request parameters

Purpose: To execute request on all lines not containing regular expression.

The following are the only legal constructions for the exclude request:

- `adr1,adr2va text/regexp/` append
- `adr1,adr2vc text/regexp/` change
- `adr1,adr2vd /regexp/` delete
- `adr1,adr2vi text/regexp/` insert
- `adr1,adr2vm bufnam/regexp/` move
- `adr1,adr2vp /regexp/` print
- `adr1,adr2vs regexp/string/regexp/` substitute
- `adr1,adr2vy string/string/` transform
- `adr1,adr2v: regexp/` absolute line
- `adr1,adr2v= /regexp/` current line

"." value: Pointer is set according to the request.

Default: vrequest request parameters is the same as 1,$vrequest request parameters

Note: Because of the nature of the exclude request, the request parameter, regexp, is required for the move request.

Example: Before:
```
  a b c d
  e f g h
  d e f h
```
Request: 1,$vp/d/
Result: e f g h
QED REQUESTS

GLOBAL g
Format: adr1,adr2 request parameters
Purpose: To execute a given request on all lines addressed.

The following are the only legal constructions for the global request:

- `adr1,adr2 ga text/regexp/` append
- `adr1,adr2 gc text/regexp/` change
- `adr1,adr2 gd /regexp/` delete
- `adr1,adr2 gi text/regexp/` insert
- `adr1,adr2 gm bufnam/regexp/` move
- `adr1,adr2 gp /regexp/` print
- `adr1,adr2 gs /regexp/string/regexp/` substitute
- `adr1,adr2 gy /string/string/` transform
- `adr1,adr2 g: /regexp/` absolute line
- `adr1,adr2 g= /regexp/` current line

"." value: Pointer is set according to the request.

Default: `request request parameters` is the same as
1,$request request parameters

Note: Because of the nature of the global request, the request parameter, `regexp`, is required for the move request.

Purpose: Before:

```
abc
def
```

Request: `1,$gp/d/

Result: `abc d
def`
**QED REQUESTS**

**INSERT**

- **Format**: \text{adr1 i text}

- **Purpose**: QED accepts text which is inserted before \text{adr1} in the current buffer.

- **"." value**: Line pointer is set to \text{adr1}.

- **Default**: \text{i text}

**Example:**

**Before:**

<table>
<thead>
<tr>
<th>BUFFER CONTENTS</th>
<th>RELATIVE ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>a: procedure;</td>
<td>1</td>
</tr>
<tr>
<td>x = y;</td>
<td>2</td>
</tr>
<tr>
<td>end a;</td>
<td>3</td>
</tr>
</tbody>
</table>

**Request:**

\text{3i}

\text{a = b;}
\text{if x = b then y = a;}
\text{end a;}

**After:**

<table>
<thead>
<tr>
<th>BUFFER CONTENTS</th>
<th>RELATIVE ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>a: procedure;</td>
<td>1</td>
</tr>
<tr>
<td>x = y;</td>
<td>2</td>
</tr>
<tr>
<td>a = b;</td>
<td>3</td>
</tr>
<tr>
<td>if x = b then y = a;</td>
<td>4</td>
</tr>
<tr>
<td>end a;</td>
<td>5</td>
</tr>
</tbody>
</table>
LIST 1

Format: 1 type segnam

Purpose: To read and print the Multics segment specified by segnam.

Only one type is currently recognized; this is:

<sp> ascii

therefore, the type parameter is left null.

"." value: is unchanged

Note: A space must appear after l.

Example:

The segment, joe.ascii, is a Multics segment in the working directory of the user.

Request: 1 joe.ascii

The contents of joe.ascii are then printed out on the console.

Result of request:

test: proc;
x = 1;
   y = x+1;
end test; } contents of joe.ascii
MOVE

Format: \texttt{adr1,adr2mx}

Purpose: To replace all the contents of buffer \texttt{x} with lines from the current buffer from \texttt{adr1} to \texttt{adr2}. \texttt{adr1} must be less than \texttt{adr2}. \texttt{adr1} through \texttt{adr2} are deleted from the current buffer. The MOVE request causes buffer \texttt{x} to become the current buffer. If buffer \texttt{x} is already the current buffer, all contents of \texttt{x} except lines specified in MOVE are deleted.

"." value: Line pointer is set to the line after the last line moved in the current buffer and set to the last line moved in buffer \texttt{x}.

Default: \texttt{adr1mx} is the same as \texttt{adr1,adr1mx}, \texttt{mx} is the same as \texttt{.mx}

Example:

Before:

\begin{verbatim}
CURRENT BUFFER L
a: proc; Line 1
  b = c; Line 2
e: proc; Line 3
  f = g; Line 4
end e; Line 5
end a; Line 6
\end{verbatim}

Request: \texttt{3,5mK}

After:

\begin{verbatim}
BUFFER L
a: proc; Line 1
  b = c; Line 2
end a; Line 3
\end{verbatim}

\begin{verbatim}
CURRENT BUFFER K
e: proc; Line 1
  f = g; Line 2
end e; Line 3
\end{verbatim}
QED REQUESTS

PRINT  p

Format:  adr1,adr2p

Purpose: To print the lines specified by adr1 through adr2. (The buffer is unchanged.)

"." value: The line pointer is set to the last line printed.

Default: adr1p is the same as adr1,adr1p

p is the same as .p
adr1 followed by a carriage return is the same as adr1p

Hitting the carriage return prints the current line.

/regular_expression/cr prints the first line in the buffer (after the current line) which contains the regular expression.

Example: Contents of Current Buffer:

    a: procedure;  Line 1
        x = y;      Line 2
        q = r;      Line 3
        s = t;      Line 4
        end a;      Line 5

Request: 2,4p or /x/,/s/p

Result:

The following is printed:

    x = y;
    q = r;
    s = t;

VI-19 Rev 2 06019
**QED REQUESTS**

**QUIT**

Format: q

Purpose: To return to Multics command level; or to return from QED to the process which called QED.

Note: The q request does not determine whether or not the buffer is saved. To save a buffer, the contents must be written into a Multics segment using either the w (WRITE) request or the optional argument in the QED command, output_file.
QED REQUESTS

READ

r

Format:  adr1r  type segnam

Purpose: To read the Multics segment whose name is specified by segnam and to append the segment after the line addressed. A space must appear between r and type.

Only one type is currently recognized.
This is:  <sp> ascii

Therefore, the type parameter may be null.

"." value: Line pointer is set to the last line read.

Default:  r type segnam is the same as $r type segnam

Example: Before:

a: procedure;  Line 1
  x = y;  Line 2
  end a;  Line 3 <-

Request:

2r joe.ascii

where joe.ascii is

b: procedure;
  c = d;
  end b;

After:

a: procedure;  Line 1
  x = y;  Line 2
b: procedure;  Line 3
  c = d;  Line 4
  end b;  Line 5 <-
  end a;  Line 6

VI-21 Rev 2 06019
SORT

Format: \( k \), \( \text{adr1,adr2}k \)

Purpose: To sort the lines specified by \( \text{adr1} \) through \( \text{adr2} \) in ascending ASCII collating sequence. \( \text{adr1} \) must be less than \( \text{adr2} \).

"." value: Line pointer is carried with the sorting; it may change value but it points to the same line of text.

Default:
- \( \text{adr1}k \) is the same as \( \text{adr1,adr1}k \)
- \( k \) is the same as \( 1,\$k \)

Example:

Current Buffer Before:
- alpha beta gamma <-
  abcde
  bcdef
  bdcef
  zxywx
  zabcd
  12345

Request: \( k \)

Current Buffer After:
- 12345
  abcde
  alpha beta gamma <-
  bcdef
  bdcef
  zabcd
  zxywx
QED REQUESTS

STATUS  x
Format: x
Purpose: To cause the following information to be listed:

- Name of current buffer.
- Value of "." (current line).
- Length of current buffer.
- Name and length of all non-zero-length auxiliary buffers.
- Names of all named regular expressions (see the Enter Request).

"." value: Line pointer is not changed.

Example: In a QED run buffers 0, 2, and 1 were mentioned in that order in BUFFER (b) requests. Regular expressions alpha and aldrich were given names. The current buffer is 1.

Request: x

Result:

  "1" 0018 0020
  "2" 0001
  "0" 0006
  alpha
  aldrich
**QED REQUESTS**

**SUBSTITUTE** s

**Format:**  
adr1,adr2s/regular expression/string/

**Purpose:**  
To replace all occurrences of an expression (regular expression) in the addressed lines with a new expression (string).

**"." value:**  
Line pointer is set to the last line substituted, or left unchanged if SUBSTITUTE finds no matching lines.

**Default:**  
adr1s/regular expression/string/  
is the same as  
adrl,adr1s/regular expression/string/

s/regular expression/string/  
is the same as  
.s/regular expression/string/

**Example:**

**Before:**

```
a: procedure;ax = y;x = z;end a;
```

Line 1  
Line 2  
Line 3  
Line 4 <-

**Request:**

```
2,3s/x/t/  or  /y/,/z/s/x/t/
```

**After:**

```
a: procedure;
t = y;
t = z; <-
end a;
```

VI-24 Rev 2 06019
TRANSFORM y

Format: adr1,adr2y/string1/string2/

Purpose: To replace occurrences of characters in string1 with the corresponding character of string2. string1 and string2 must be of the same length; no character may appear twice in string1.

"." value: Set to the last line transformed.

Default: adr1y/string1/string2/

is the same as

adr1,adr1y/string1/string2/

y/string1/string2/

is the same as

.y/string1/string2/

Example: Current Buffer Before:

AAAAAAA
Aardvaark
ABA
ABAFT
ABB
ABBACY

Request:

1,$y/ABC/abc/

Current Buffer After:

aaaaaaa
aardvaark
aba
abaFT
abb
abbacy <-

VI-25 Rev 2 06019
QED REQUESTS

WRITE  w
Format:  adr1,adr2w type segnam
Purpose: To write the addressed lines into the segment <segnam>.

Only one type is recognized. This is:

<sp> ascii,

therefore, the type parameter may be null.

A space must appear between w and type (or segnam).

"," value: Line pointer is unchanged.

Default: w type segnam

is the same as

1,$w type segnam

Example:

Request: 2,3w sam.ascii

Result:

The second and third lines of the current buffer are written into the segment sam.ascii in the user's working directory.
Probe requests perform the following functions:

1. Direct output from probe to a standard stream (console), or direct output to a user specified segment.
2. Dump machine conditions and register contents.
3. Dump all or part of a segment.
4. Dump the contents of an entire process directory.
5. Print size, access, and date of creation information for one segment or a group of segments.
6. Print a stack trace for a process.
7. Print argument list for a stack frame.
8. Make a segment known or unknown to the system.
9. Print a summary of available probe requests.
10. Make an octal patch to a segment.
PROBE REQUESTS

ARGLIST

Format: arglist stack frame

Purpose: To print an argument list for the specified stack frame.

stack is the name or number of the stack segment

frame is the name of the "owning procedure" or starting offset of a stack frame.

If stack is not given, the current stack is assumed; the argument list for the last occurrence of frame in the current stack is printed.

Example1: arglist 4760

Might cause the following to be printed for frame 4760 in the current stack:

arg_1 fixed, bin
26
arg_2 varying character string
"no_comment_necessary"
arg_3 bit string
1260

Example2: arglist stack_00 gim

The example presumes operation in a ring other than ring 0. Therefore, the ring 0 stack is given, followed by the requested frame.
PROBE REQUESTS

DUMP_PROCESS

Format: dump_process process_id

Purpose: To obtain an octal dump of each segment in the process whose unique identifier is given by process_id. The unique process_id may be given in octal or as a character string. If no argument is given, an octal dump of the current user process will result.

Note: Normally, output will be directed by the use of the output request to a segment for later printing.

Example: - dump_process
PROBE REQUESTS

INFO

Format: info

Purpose: To provide a complete list of probe requests with pertinent parameters and options and provide an abbreviated explanation of the request's use.

Example: - info

See BX.10.00A for complete descriptions.
The following requests are available:

arglist
info
initiate xxx
output p -xxx-
segdump xxx
segments -xxx- -xxx-
segstatus xxx
stack -xxx- -yyy-
terminate
quit

- list of stack frames
- obtain this listing
- make a segment known to process
- direct to a specific medium
- part of a segment in octal
- print information about a group of segments
- print information about one segment
- print a stack trace of segment starting with frame yyy
- make a segment unknown to process
- return to command level

Note: The current info printout shown above is not complete and will probably change shortly.
INITIATE

Format: initiate path reference

Purpose: To make the segment given by path known to the process being debugged by the reference name given by reference. If reference is not given, the entry name of the segment will be used.

Example:
Request: initiate bin_oct
Response: Segment bin_oct initiated. Number 41
PROBE REQUESTS

OUTPUT

Format: output console
output segment path

Purpose: output console - directs output from probe request to the console.
output segment path - directs output to a segment whose path-name is path.

Note: By default, output of probe requests is printed on the console. The output console request need only be issued after a previous output segment path.

Example: probe
W 924:09.4
- output segment text_prog (request)
Output directed to segment text_prog.
Number 227 (response)
- state (The output of probe request, state, is to text_prog in the user's working directory.)
- output console (Subsequent probe request to redirect output to the console.)
QUIT

Format: quit

Purpose: To stop processing probe requests and return to Multics command level.

Example:
- quit
  r 5:04.0 19.2 40
SEGDRVMP

Format:  segdump seg lower upper

Purpose: To produce an octal dump of the segment
          seg from the lower bound specified by
          lower to the upper bound specified by
          upper.  seg is the segment name or an
          octal number which designates a segment
          in the KST.

          If the parameter upper is not specified,
          the segment is dumped from lower to the
          current length of the segment.

          If neither parameter is specified, the
          entire segment is dumped.

Note:  If the segment is not known to the
        process, the comment: segment not yet
        initiated is printed.

Example:

        segdump 203 1700 1777
        Segment multics 000203
        001700  000114352000  600556350100  600556757100  600562252100
        001710  000560252100  000114352000  600556757100  600562252100
        001720  000222352000  600562252100  000114352000  600556757100
        001730  400064710110  000556350100  600562252100  000114352000
        001740  000242352000  000114352000  600556252100  000114352000
        001750  000560252100  000114352000  600556252100  000114352000
        001760  000232352000  000114352000  600556252100  000114352000
        001770  400070710120  000114352000  600556252100  000114352000

VII-8 Rev 2 06019
SEGINFO

Format:    seginfo seg1 seg2 all long
Purpose:   To print a list of segment names and numbers known to the process from segment seg1 through segment seg2.

seg1 and seg2 are either segment names or numbers known to the process being debugged. If seg2 is blank, a list of all segments from seg1 are printed. If both seg1 and seg2 are blank, names of all segments known to the process are printed. Source of information for the list is the KST.

all is an optional literal causing all the reference names for each listed segment to be printed. (A reference name is a name in the KST by which a segment is known to a process).

long is an optional literal causing the following information to be printed for each listed segment: current length, access modes and date created.

If a request is:    seginfo
a list of all segments known to the process being debugged is printed.

Any combination of parameters to the seginfo request is permissible, except use of seg2 without a preceding seg1.

Example:    seginfo 200 test_proc long
Results:

200  root>sys_root>sys_lib>cv_string.link  1 rewa 12/31/68 1900.0 EST MON
201  root>sys_root>sys_lib>get_count  8 re 12/31/68 1900.0 EST MON
202  root>sys_root>sys_lib>sys_info.link  1 re 12/31/68 1900.0 EST MON
PROBE REQUESTS

SET

Format: set seg location value1 value2 ....

Purpose: To place values beginning with value1 in the segment given by seg, beginning at the location given by location. value1 is placed in location, value2 in location+1, etc. At least one value must be present. The request allows octal patching of segments for which the user has write permission.

Notes: seg may be a symbolic name or segment number.

Example: set 2031700 000224251000 600525210000 where the values given replace the current values in locations 001700 and 001701 of the segment numbered 203. Probe prints out the values before and after the change.
STACK

Format: stack seg frame f args

Purpose: To trace the sequence of calls in stack segment seg starting at location frame. seg may be a segment number or name. If frame is given in octal, it is interpreted as a frame number. If frame is given as a segment name, the stack is examined for a frame belonging to the segment. Tracing starts at that frame.

Default tracing is from end to beginning of the stack through ring-crossing frames. Optional literal f causes tracing to proceed from beginning to end, terminating when a ring-crossing frame is encountered.

If neither seg nor frame are given, the current stack is assumed. seg must be given if frame is given. If frame is not given, tracing proceeds from either the beginning or end of stack as appropriate.

Optional literal args causes a list of all arguments passed to each stack frame to be printed.

For each frame, the name and number of the segment using the frame, starting location in the stack segment, and frame size are printed.

Example: - stack

stack trace of segment stack_01. Number 000171.

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Start</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>0226</td>
<td>probe</td>
<td>1314</td>
<td>003540</td>
</tr>
<tr>
<td>0226</td>
<td>probe</td>
<td>223</td>
<td>003410</td>
</tr>
<tr>
<td>0225</td>
<td>shell_char</td>
<td>4242</td>
<td>002150</td>
</tr>
<tr>
<td>0225</td>
<td>shell_char</td>
<td>1075</td>
<td>001610</td>
</tr>
<tr>
<td>0225</td>
<td>signal</td>
<td>464</td>
<td>001340</td>
</tr>
<tr>
<td>0011</td>
<td>fim</td>
<td>56</td>
<td>001160</td>
</tr>
<tr>
<td>0203</td>
<td>multics</td>
<td>1747</td>
<td>000310</td>
</tr>
<tr>
<td>0206</td>
<td>bit_to</td>
<td>2147</td>
<td>000220</td>
</tr>
<tr>
<td>0203</td>
<td>multics</td>
<td>3153</td>
<td>000050</td>
</tr>
<tr>
<td>0000</td>
<td>NOCALL</td>
<td>0</td>
<td>000010</td>
</tr>
</tbody>
</table>
**STATE**

**Formats:**

<table>
<thead>
<tr>
<th>Request</th>
<th>Meaning of Request</th>
</tr>
</thead>
<tbody>
<tr>
<td>state arith</td>
<td>Print contents of A, Q, and exponent registers.</td>
</tr>
<tr>
<td>state bases</td>
<td>Print contents of 8 base registers.</td>
</tr>
<tr>
<td>state cunit</td>
<td>Print control unit contents and ring number.</td>
</tr>
<tr>
<td>state index</td>
<td>Print contents of 8 index registers.</td>
</tr>
<tr>
<td>state location</td>
<td>Print the fault location and the computed address.</td>
</tr>
<tr>
<td>state timer</td>
<td>Print contents of timer register.</td>
</tr>
<tr>
<td>state</td>
<td>Print all of the above.</td>
</tr>
</tbody>
</table>

**Purpose:** To print the available status information for the process being debugged.

**Example:** `-state`

\[
\begin{array}{ll}
A: & 000004000000 \\
Q: & 000000000000 \\
Exponent: & 000000000000 \\
Indicator: & 10 \\
Timer: & 430351270000 \\
Fault at: & 20311746 \\
Effective address: & 201162 \\
Index registers: & 3000171041066 \\
Base registers: & \\
ap: & 001066100000 \\
ab: & 000171040000 \\
bp: & 000242300000 \\
bb: & 000203040000 \\
lp: & 000072500000 \\
lb: & 000201040000 \\
sp: & 000310700000 \\
sb: & 000171060000 \\
Control unit: & \\
000201022001 & 000162000200 & 000203200700 & 001746001000 \\
000162710120 & 000232352000 \\
Ring: & 001 \\
\end{array}
\]
PROBE REQUESTS

TERMINATE

Format:  terminate path

Purpose:  To make the segment given by path unknown to the process by removing it from the KST.

Example:  terminate bin_oct