Subject: FAST - a Multics oriented subsytem

Date: 1/26/76

Author: Susan Barr

Introduction

The DFAST subsystem (previously called FAST in MTB-202) provides a simple transition for users with a background on DTSS (Dartmouth Time-Sharing System). The FAST subsystem proposed here is aimed at users with little computer experience. It emphasizes simplicity and compatibility with Multics.

The main features of the command interface are described here. A future MTB will dicuss issues of the runtime envircnment. A great deal of thought, discussion, and effort has gone into the top-level specification of the user interface for FAST: it is believed that the interface specified here represents the best compromise for the design objectives. However, suggestions for changes at the detailed level to this version or possible improvements for future releases would be appreciated and should be sent by February 16 to me (Barr.Multics on Multics at MIT or Honeywell Information Systems, 575 Tech. Sq., Cambridge).

Goals

In the design of the user interface of FAST we have tried to limit the number of commands and functions provided. This is not considered a restriction since a user with a special requirement (tape input/output, for example) can use the full Multics system. The aims of the subsystem are:

1. Ease of Use: The user interface should be simple to learn, remember and use.
2. Performance: The implementation must emphasize efficiency and high performance.
3. Compatibility with Multics: It should be almost effortless for a user to switch between full Multics and this new limited environment.

Overview of FAST

FAST is implemented as a process overseer. It has an integrated line numbered editor and command processor. The user can create and modify text from command level. Although this feature is not compatible with Multics, it is felt that the ease of use of this mechanism for beginners and casual users makes it necessary.

In general, FAST is a subset of Multics; features have been restricted, but not changed. For example, the Multics erase, kill and canonicalization conventions will be supported. The following list describes the major restrictions:

1. Command processor:

   The restrictions are similar to the checks made by the first pass of the Multics command processor.

   a. There is a maximum of 10 arguments allowed for a command.

b. No iteration or quote processing is performed.

c. Active functions are not available.

d. Only a fixed list of commands is available.

2. QUIT signal:

When the user signals QUIT, FAST will take one of two actions:

a. If the command being executed is edm, the system will query, "Do you want to continue editing ?". If the user responds "yes", the program_interrupt condition will be signalled. Otherwise a release will be done.

b. If the command is not edm, the release will be done.

3. Error conditions:

FAST will have an any_other handler. It will call condition_interpreter_ to print a message and ther do a release.

4. Segment names:

The characters allowed in pathnames will be restricted so that users won't create segments that have names with non-printing characters and other characters that would conflict with star convention etc. This is similar to the check made in edm.

Working Environment

The user's working environment provides a direct method of creating and editing line-numbered files, a set of edit commands to modify file content, a set of commands to manipulate file attributes, and a set of commands to compile and run programs. The edit commands modify a temporary segment, the contents of which are referred to as the temporary text. The user can edit an existing segment by using the "old" command to copy a segment into this temporary segment. A program can be modified and executed without changing the original copy.

A line that begins with a digit is assumed to be text. All other lines are assumed to be commands. Text lines are stored in ascending order by line number. It is not possible to edit text that does not contain line numbers at command level. (The command edm is available for use with text without line numbers).

The "run" command can be used to compile and execute soucre code or to execute object segments.

The following document is a rough draft of the commands section of the FAST Users' Guide. All of the commands available under FAST are listed in the summary. All of the new commands for FAST are described in detailed descriptions following the summary.

The section that gives the detailed descriptions of each command is incomplete. While descriptions of all of the new commands for FAST are given, most of the Multics commands are not yet included. The final version of the Users' Guide will have a description of every command. The FAST Users' Guide should contain all information necessary to use the subsystem.

In general, a subset of the options for each Multics command will be documented even though the user is not restricted to that subset. This is being done to make learning the command repetoire easier.

SECTION III

COMMAND OVERVIEW

Command Lines

A line containing a single command can begin at any horizontal position. When arguments are supplied, at least one blank or tab must separate them from the command. Arguments are separated from each other by blanks or tabs and the entire line is terminated by a new line.

Any line that begins with a number is interpreted as a line of input text. Preceding blanks or tabs are ignored. All of the following lines will be entered into the current segment.

```
100 if x <= y then 120
110 if x < z then let x = x + 1.2
5   data 12, 20, 35
    7 end
```

Line numbers can range from 1 to 99999. Lines can be entered in any order. They are automatically sorted into ascending line number sequence. If the user types in a line with a number that has been entered previously, the new text replaces the old associated with that line number. If a user types in a line number with no text, the existing line with that number is deleted.

Segment Naming Conventions

A segment name is a user-constructed identifier from one to 32 characters long. It can contain any uppercase or lowercase alphabetic character, any number (0-9), and the characters hyphen (-), underscore (_), and period (.). A period has a special effect, dividing a user construct into separate components to be interpreted by the system. For example, the use of the period in:

```
     test.fortran
```

produces a two-component name whose second component is a language suffix indicating that the segment is a FORTRAN source program.

FORTRAN and BASIC files should follow the same naming conventions as segments.

FAST COMMAND REPERTOIRE

The complete list of FAST commands is given below, organized in terms of general function. A detailed description of each of these commands is provided in Section IV. Many of the commands are subsets of the standard Multics commands. In those cases, only the arguments of general interest have been documented. The number given in parenthesis is the level of difficulty.

level 1	These commands are the easiest to learn and to use. The seven level 1 commands are the minimum required to create and run programs.
level 2	These commands allow the user to create object segments, print information about the system and do more complicated editing.
level 3	These commands allow the user to set access on segments and to use programs and segments not in the user's working directory.

Access to the System

(1)	login	connects registered user to the system; used at dialup.
(1)	logout	terminates a user session and disconnects the terminal.

Edit and Print

(1)	new	deletes the temporary text and changes the default name.
(1)	old	replaces the temporary text with the contents of a previously saved segment.
(1)	print_text	prints all or portions of the temporary text.
(1)	save	copies the temporary text into the segment specified.
(2)	change	replaces a specified character string within a line.
(2)	delete_line_numbers	removes the line number from each line of the text.

(2)	delete_text	deletes lines from the temporary text.
(2)	locate	prints lines from the temporary text containing a specified string.
(2)	input	establishes a mode of input where the system supplies the line number and the user completes the line.
(2)	merge	inserts the contents of a segment into the temporary text.
(2)	move_text	relocates one or more lines of the temporary text.
(2)	resequence	changes the line numbers in the text.
(2)	add_line_numbers	adds a line number to each line of the text.
(3)	edm	invokes an editor for use with text without line numbers.
(3)	dprint	queues a segment for printing on the high-speed line printer.
(3)	dpunch	queues a segment for card punching.

Compile and execute

(1)	run	compiles and executes the temporary text or executes object code in a segment.
(2)	basic	creates a basic object segment.
(2)	fortran	creates a fortran object segment.

Information

(2)	help	prints on-line description of specified topic.
(2)	info	prints the default name, date, time, quota, money spent and total money allotted
(2)	hmu	prints the number of users.

(2)	ready_off	suppresses the prompt character.
(2)	ready_on	causes the prompt character to be printed

Storage System

(2)	copy	copies a segment.
(2)	list	prints information about segments.
(2)	delete	deletes a segment.
(3)	add_name	adds a rame to a segment.
(3)	delete_name	deletes a name from a segment.
(3)	link	creates a link.
(3)	unlink	deletes a link

Access Control

(3)	delete_acl	removes an ACL entry.
(3)	list_acl	prints an ACL entry.
(3)	set_acl	adds or changes an ACL entry.

Terminal Control

(3)	set_tty	allows the user to change the defaults for the terminal.

SECTION IV

COMMAND DESCRIPTIONS

This section contains, in alphabetical order, a description of each of the FAST commands giving its usage and function and illustrating its application in a user session. The contents and notation conventions associated with the various divisions of a command description are given below.

NAME

The heading, "Name:" is followed by the full command name which in turn is followed by a comma and the valid abbreviation for the command, as in:

Name: list, ls

Here, the list command can be invoked by typing either "list" or "ls".

USAGE

The heading "Usage" is followed by a line showing a prototype command line. Optional arguments are enclosed by braces, as in:

```
save {segment name}
```

Here, segment_name is an optional argument and valid user-supplied entries for it are given after the format line. Arguments are shown in the order in which they should be supplied. Required arguments appear without surrounding braces.

EXAMPLE

Under the heading "Example", portions of user-FAST dialogue are given to show the usage and effects of executing the command. In these dialogues, the user's typing is preceded by an exclamation point (!). This is purely a notational convention and should not be typed by the user in an actual session.

Name: add_line_numbers, adl

The add_line_numbers command adds a new set of line numbers to a segment specified by the pathname given in the command line. If no pathname is given the temporary buffer is used.

Usage

```
     add_line_numbers  path  {new_number {increment}}
```

where:

1.	path	is the pathname of the segment to be modified.
2.	new_number	is the first line number to be added (100 by default).
3.	increment	is the increment used to derive subsequent numbers (10 by default).

Example

```
!    print_text data_1

     data_1  01/12/76  1539 est Mon

     non-numbered
     data segment
     input
     r

!    add_line_numbers data_1                /default values are used
     r

!    print_text data_1 -nhe
     100 non-numbered
     110 data segment
     120 input
     r

!    add_line_numbers data_1 500 5
     r

!    print_text data_1  -nhe
     500 100 non-numbered
     505 110 data segment
```

```
510 120 input
r
```

Notes

The value of new_number is used for the first line and the increment is added to derive subsequent numbers. If the text already has line numbers, these are retained but become part of the text on the line. If no increment is supplied, ten is assumed. If no arguments are supplied, the first line number in the file will be 100.

Name: change, c

The change command replaces a string of characters within a line with a new string. The change request can apply to one line or a range of lines. It is not possible to change the line number at the beginning of the line with this command.

Usage

```
     change  /old_string/new_string/  first_line  {last_line}
```

where:

1.	/	is any delimitor except blank, tab or a digit.
2.	old_string	is a string of characters to be replaced.
3.	new_string	is a string of characters to be substituted for each occurrence of old_string.
4.	first_line	is the first line to be changed.
5.	last_line	is the last_line to be changed; if this argument is not given the change is only made for first_line.

Example

```
!     130 for n = 1 to 5
!     140 let e = 40
!     150 for m = 1 to 3
!     160 let e = e + p(m)
!     change /e/s/ 150 170
      r

!     print_text -nne
!     130 for n = 1 to 5
!     140 let e = 40
!     150 for m = 1 to 3
!     160 lst s = s + p(m)      /"e" in "let" was changed
      r
```

```
!     change  /lst/let/ 160
      r

!     print_text  160
      160 let s = s + p(m)
      r
```

Name: delete, dl

The delete command removes a segment from the user's working directory or from another directory, if specified in the path argument. A delete can only be successful if the user has appropriate access to the segment specified.

Usage

```
     delete path
```

where path is the absolute or relative pathname of a segment.

Example

```
!    delete test.basic
     r
```

Name: delete_acl, da

The delete_acl command removes entries from the access control lists (ACLs) of segments.

Usage

```
     delete_acl {path} {User_ids} {-control_args}
```

where:

1.	path	is the pathname of a segment.
2.	User_ids	are access control names that must be of the form Person_id.Project_id.tag. All ACL entries with matching names are deleted. (For a description of the matching strategy, refer to the set_acl command.) If User_id is -a or -all, the entire ACL is deleted with the exception of an entry for *.SysDaemon.*.
3.	control_args	can be chosen from the following:
	-all, -a	causes the entire ACL to be deleted with the exception of an entry for *.SysDaemon.*.
	-brief, -bf	suppresses the message "User name not on ACL."

Examples

```
     delete_acl news .Faculty. Jones
```

deletes from the ACL of news all entries with Project_id Faculty and the entry for Jones.*.*.

```
     da beta.** ..
```

deletes from the ACL of every segment whose entryname has a first component of beta all entries except the one for *.SysDaemon.*.

Notes

If the delete_acl command is invoked with no arguments, it deletes the entry for the user's Person_id and current Project_id on the ACL of the working directory. (Usually the user does not have access to do this.)

An ACL entry for *.SysDaemon.* can be deleted only by specifying all three components. The user should be aware that in deleting access to the SysDaemon project he prevents Backup.SysDaemon.* from saving the segment or directory (including the hierarchy inferior to the directory) on tape, Dumper.SysDaemon.* from reloading it, and Retriever.SysDaemon.* from retrieving it.

The user needs modify permission on the containing directory.

Name: delete_line_numbers, dln

The delete_line_numbers command removes the line number and one blank following it from each line of a segment. If a line is found without a line number, it is left unchanged.

Usage

```
     delete_line_numbers  path
```

where:

path	is the pathname of the segment to be modified.

Example

```
!    print_text  data

     data  01/12/76  1254.0 est Mon

!    10 ten
!    20 twenty
!    30  thirty

     r

!    delete_line_numbers data
     r

!    print_text data

     data  01/12/76  1539 est Mon

     ten
     twenty
      thirty

     r
```

Note

It is not possible to use the command level editor on text without line numbers. Unnumbered text can be modified using the edm command or new line numbers can be added with the add_line_numbers command.

Name: delete_text, dt

The delete_text command deletes one or more lines of the temporary text.

Usage

```
     delete_text first_line  {last_line}
```

where:

1.	first_line	is the line number of the first line to be deleted.
2.	last_line	is the line number of the last line to be deleted. If this number is not given, only first_line will be deleted.

Example

```
!    print_text

     eval.basic  1/16/76  1520 est Fri

     100 input n
     110 for i = 1 to n
     120 input x
     130 let t1 = t1 + x
     140 let t2 = t2 + y
     150 if t2 > 9000 then 500
     160 next i


     r

!    delete_text 140 150
     r

!    print_text -nhe
     100 input n
     110 for i = 1 to n
     120 input x
     130 let t1 = t1 + x
     160 next i
     r
```

__Name__: dprint, dp

The dprint command queues specified segments for printing on the line printer. The output is identified by the requestor's User_id.

__Usage__

```
     dprint {-control_args} {paths}
```

where:

1. control_args	may be chosen from the following list of control arguments and can appear anywhere in the command line:
-copy _n_, -cp _n_	prints _n_ copies ($n \leq 4$) of specified paths. This control argument can be overruled by a subsequent -copy control argument. If path_i_ is to be deleted after printing, all _n_ copies are printed first. If this control argument is not given, one copy is made.
-queue _n_, -q _n_	prints specified paths in priority queue _n_ ($n \leq 3$). This control argument can be overruled by a subsequent -queue control argument. If this control argument is not given, queue 3 is assumed. (See "Notes" below.)
-header XX, -he XX	identifies subsequent output by the string XX. If this control argument is not given, the default is the requestor's Person_id. This argument can be overruled by a subsequent -header control argument.

-destination XX, -ds XX	labels subsequent output with the string XX, which is used to determine where to deliver the output. If this control argument is not giver, the default is the requestor's Project_id. This argument can be overruled by a subsequent -destination control argument.
2. paths	are the names of segments to be queued for printing.

Notes

If the dprint command is invoked without any arguments, the system prints a message giving the status of queue 3.

If control arguments are present, they affect only paths specified after their appearance in the command line. If control arguments are given without a following path_ argument, they are ignored for this invocation of the command and a warning message is printed.

The -queue 1 control argument places requests in the top priority queue, -queue 2 places them in the second priority queue, and -queue 3 (or not specifying a queue) places them in the lowest priority queue. All requests in the first queue are processed before any requests in the other queues, and so on. Higher priority queues usually have a higher cost associated with them.

Paths cannot be printed unless appropriate system processes have sufficient access. The process that runs devices of the specified class (normally IO.SysDaemon) must have read access to all paths to be printed and status permission on the containing directory.

Example

The command:

```
!     dp -he Jones -cp 2  test.basic test.fortran
```

causes two copies of each of the segments named test.basic and test.fortran in the working directory to be printed with the header "Jones".

Name: dpunch, dpn

The dpunch command queues specified segments and files for punching by the card punch. It is similar to the dprint command.

Usage

```
dpunch {-control_args} {paths}
```

where:

1. control_args	may be chosen from the following list of control arguments and can appear anywhere in the command line after the command:
-copy *n*, -cp *n*	punches *n* copies ($n \leq 4$) of all specified paths. This control argument can be overruled by a subsequent -copy control argument. If path*i* is to be deleted after punching, all *n* copies are punched first. If this control argument is not given, one copy is made.
-queue *n*, -q *n*	punches specified paths in priority queue *n* ($n \leq 3$). This control argument can be overruled by a subsequent -queue control argument. If this control argument is not given, queue 3 is assumed. (See "Notes" below.)
-header XX, -he XX	identifies subsequent output by the string XX. If this control argument is not given, the default is the requestor's Person_id. This control argument can be overruled by a subsequent -header control argument.
-destination XX -ds XX	uses the string XX to determine where to deliver the deck. If this control argument is not given, the default is the requestor's Project_id. This control argument can be overruled by a subsequent -destination control argument.

-mcc	punches the specified paths in the command line using character conversion. This control argument can be overruled by either the -raw or -7punch control arguments.
-raw	punches the specified paths in the command line using no conversion. This control argument can be overruled by either the -mcc or -7punch control arguments.
-7punch, -7p	punches the specified paths in the command line using 7-punch conversion. This is the default conversion mode and need only be specified when a number of segments are being requested by one invocation of dpunch and other modes (-mcc or -raw) have been specified earlier in the command line. For a description of conversion modes, see "Bulk Input/Output" in Section IV of the MPM Reference Guide.
2. paths	are the names of files to be queued for punching.

Notes

If the dpunch command is invoked without any arguments, the system prints a message giving the status of queue 3.

If control arguments are present, they affect only paths specified after their appearance on the command line. If control arguments are given without a following path*i* argument, they are ignored for this invocation of the command and a warning message is returned.

The -queue 1 control argument places requests in the top priority queue, -queue 2 places them in the second priority queue, and -queue 3 (or not specifying a queue) places them in the lowest priority queue. All requests in the first queue are processed before any requests in the other queues, and so on. Higher priority queues usually have a higher cost associated with them.

A path*i* cannot be punched unless appropriate system processes have sufficient access. The process (normally IO.SysDaemon) that runs devices of the specified class must have read access to all paths to be punched and status permission on the containing directory.

The dpunch command does not accept the star convention; it prints a warning message if a name containing asterisks is encountered and continues processing its other arguments.

Example

The command:

```
dpunch -he Smith test.fortran
```

causes the file named test.fortran to be punched using 7-punch conversion (the default conversion mode) with "for Smith" added to the heading.

Name: fortran, ft

The fortran command invokes the FORTRAN compiler to translate a segment containing the text of a FORTRAN source program into a object segment. The object segment is saved in the user's working directory.

Usage

```
     fortran  pathname  {-control_arg}
```

where:

1.	pathname	is the pathname of the source program. If the path does not have a suffix of ".fortran" then one is assumed. However, the suffix ".fortran" must be the last component of the name of the segment.
2.	-control_arg	is -no_line_numbers or -nn.
	-no_line_numbers, -nn	specifies text without line numbers

Example

```
!     fortran mpg.fortran
      r


!     fortran test                 /test.fortran is compiled
      r
```

Name: help

The help command prints a specific online description maintained by the subsystem. Such a description is maintained for each command and for general topics such as file access. A list of topics available can be printed by issuing the command with "topics" as its argument.

Usage

```
     help {topic1...topicn}
```

where:

topic	is a keyword indicating the help message desired.

Examples

```
!    help new

     Usage:          new  [file_rame]

          The new command truncates the temporary buffer.  If
     the name is not given with the new command, it is requested.

     r


!    help teach
     help:  no info segment for "teach"
     r
```

Name: how_many_users, hmu

The how_many_users command prints the number of users currently logged in under Multics.

Usage

```
     how_many_users
```

Example

```
!    how_many_users


     Multics MR3.1, load 18.0/110.0; 18 users
     Absentee users 0/30


     r
```

Name: input

The input command provides a convient way to enter rumbered lines. The system types the line number and one space. The user completes the line. This input mode is terminated when the user types carriage return without having typed any other characters on the line.

If the number given with the input command or a number generated by the input request already exist (or would cause the temporary text to be out of sequence), an error message is printed and the user returns to command level.

Usage

```
     input  {new_number}  {increment}
```

where:

1. new_number	is the number of the first new line. If this number is not given lines are entered at the end of the temporary text.
2. increment	is the increment to use to derive subsequent line numbers. (10 by default)

Example

```
!    new test.basic
     r

!    input
     00100 !let a = 10.2
     00110 !let x = 3.9
     00120 !
     r

!    print_text

     test.basic  01/22/76 1507 ext Fri

     00100 let a = 10.2
     00110 let x = 3.9
     r
```

Name: list, ls

When a segment is saved, its name and other information about it is placed in the directory specified (by default, the user's working directory) and forms a unit called an entry. To list information maintained in a directory, the user can issue a list command with a variety of control arguments that are used to restrict the listing to a subset of entries and/or a subset of information. When no arguments are given, the listing gives the name, access mode and length for each entry in the working directory in the order in which they were created.

Usage

```
     list {entrynames} {-control_args}
```

where:

1. entrynames — are a subset of entries to be considered for listing. Listing of information about these entries depends on the control arguments given.

2. control_args — may be chosen from the arguments given below and supplied in any order. Arguments are organized below under functional category, where appropriate. The arguments themselves are identified by the character hyphen (-) which must immediately precede them. The basic output format of list is a series of columns, each of which corresponds to an attribute of the entry. If no attributes are explicitly stated, name, access mode and records used are printed. Otherwise, only the name and specified attributes are printed. Both totals and detailed information are printed unless the user specifies otherwise. Entries

are printed in the order they occur unless the user explicitly requests a different order.

-pathname path, -pn path	specifies a directory to be listed; if not supplied, the working directory is assumed.
-name, -nm	prints the names column.
-date_time_modified, -dtm	prints the date and time the entry was last modified.
-date_time_used, -dtu	prints the date and time the entry was last used.
-total, -tt	prints only the heading line for each entry type, giving the total number of entries and the total number of records used.
-no_header, -nhe	omits all heading lines.

Examples

```
!     list

      Segments= 4, Records= 26.

      r w   10  test.basic
      rew    9  test
      r w    5  newfile
      r w    2  summary.basic

      r

!     list *.basic

      Segments= 2, Records= 12.

      r w   10  test.basic
      r w    2  summary.basic

      r
```

Name: list_acl, la

The list_acl command lists the access control lists (ACLs) of segments and directories. (See "Access Control" in Section III.)

Usage

```
list_acl {path} {User_ids} {-control_args}
```

where:

1. path	is the pathname of a segment or directory. If it is -wd, -working_directory, or omitted, the working directory is assumed. If it is omitted, no User_ids can be specified. The star convention can be used.
2. User_ids	are access control names that must be of the form Person_id.Project_id.tag. All ACL entries with matching names are listed. (For a description of the matching strategy, refer to the set_acl command.) If User_id is -a, -all, or omitted, the entire ACL is listed.
3. control_args	can be chosen from the following control arguments:
-all, -a	lists the entire ACL. This argument overrides any specified User_ids.
-brief, -bf	suppresses the message "User name not on ACL of path."
-directory, -dr	lists the ACLs of directories only. The default is segments and directories.

Note

If the list_acl command is invoked with no arguments, it lists the entire ACL of the working directory.

Examples

```
     list_acl notice.runoff .Faculty. Doe
```

lists, from the ACL of notice.runoff, all entries with Project_id Faculty and the entry for Doe.*.*.

```
     list_acl *.basic
```

lists the whole ACL of every segment in the working directory that has a two-component name with a second component of basic.

```
     la -wd .Faculty. *.*.*
```

lists access modes for all entries on the working directory's ACL whose middle component is Faculty and for the *.*.* entry.

Name: locate, l

The locate command causes the temporary text to be searched for all occurrences of a specified character string. Each line containing the string is printed. The entire line will be used in matching the the string including the line number.

Usage

```
locate /string/  {first_line}  {last_line}
```

where:

1.	/	is any delimitor except blank, tab or a digit.
2.	old_string	is a string of characters to be found.
3.	first_line	is the first line to be searched. If this line is missing the entire text will be searched.
3.	last_line	is the last_line to be searched; if this argument is not given the search is made from first_line to the end of the text.

Example

```
!    130 for n = 1 to 5
!    140 let e = 40
!    150 for m = 1 to 3
!    160 let e = e + p(m)
!    locate /m/                 /entire text is searched
     150 for m = 1 to 3
     160 let e = e + p(m)
     r
```

Name: logout

The logout command terminates a user session and ends communication with the FAST system.

On terminals equipped with acoustic couplers, it is necessary to hang up the telephone handset. In this case, the additional message, "hangup", is printed.

Usage

```
     logout {-control_arg-}
```

where control_arg can be -hold. If this argument is given, the user's session is terminated. However, communication with the Multics system is not terminated, and a user can immediately log in without redialing.

Examples

```
!    logout


     Smith Design logged out 11/07/75  1240.4 mst Fri
     CPU usage 5 sec, memory usage 16.5 units.


!    logout -hold


     Smith Design logged out 11/07/75  1240.4 mst Fri
     CPU usage 5 sec, memory usage 16.5 units.

     Multics 3.1: PCO, Pheonix, Az.
     Load = 41.0 out of 80.0 units: users = 41
!    l Jones                      /new user
     (etc.)
```

Name: merge

The merge command inserts the contents of a segment into the temporary text. The text to be inserted is resequeced so that the new text will not duplicate any existing line numbers.

Special editing is done for BASIC source text. Any references to the lines that were renumbered are edited to reflect the new numbers. This editing is done only if the last name given with the new, old, or save commands ends with ".basic".

Usage

```
     merge  path  line_number
```

where:

1.	path	specifies the segment to be inserted.
3.	line_number	specifies the line after which the segment will be inserted.

Example

```
!    print_text check          /segment to be inserted

     check  01/17/76  1502 est FRI

     100 if x > y then 150
     120 let w = y
     130 let y = x
     140 let x = w
     150 call "trans": x,y
     r

!    print_text

     test.basic 01/17/76  1502 est Fri

     10 input x,y,z
     12 goto 10
     r

!    merge check 10
     r
```

```
!    print_text -nhe
     10 input x,y,z
     20 if x > y then 60
     30 let w = y
     40 let y = x
     50 let x = w
     60 call "trans": x,y
     70 goto 10                    /this number was changed to prevent
                                   /overlap

     r
```

<u>Note</u>

The segment specified with this command must have line numbers.

Name: move_text, mt

The move_text command relocates one or more lines of the temporary text. The lines that are moved are resequenced. If the new line numbers would cause duplication of existing line numbers, enough lines of the text will be resequenced to insure no overlap.

Special editing is done for BASIC source text. Any references to the lines that were renumbered are edited to reflect the new numbers. This editing is done only if the last name given with the new, old, or save commands ends with ".basic".

Usage

```
     move_text  first_line  [last_line]  ,line_number
```

where:

1.	first_line	is the line number of the first line to be move.
2.	last_line	is the line number of the last line to be moved. If the argument is not given, only first_line will be moved.
3.	line_number	specifies the line after which the the moved lines of lines will be inserted.

Example

```
!    print_text

     test.basic  01/17/76  1502 est FRI

     100 if x > m then 160
     110 if x < 0 then 140
     120 let t = t + x
     130 goto 100
     140 print "illegal x"
     150 stop
     160 gosub 300

     r
```

```
!     move_text 140 155, 600
      r

!     print_text -nhe
      100 if x > m then 160
      110 if x < 0 then 610      /140 changed to 610
      120 let t = t + x
      130 goto 100
      160 gosub 300
      610 print "illegal x"      /location following line 600
      620 stop
      r
```

Note

The first_line and last_line do not have to appear in the text, but the range specified by them must contain at least one line.

Name: new

The new command truncates the temporary buffer. If a pathname is not given with the new command it is requested.

Usage

```
     new {path}
```

where:

path	is the pathname for the segment being created. (See "Segment Naming Conventions" in Section III for a description of valid segment names.)

Examples

```
!    new
     enter name:  !   newfile.basic
     r


!    new variance.fortran
     r
```

Note:

The pathname given with the new command will be used with the save command and will be printed with the lines and info commands.

Name: old

The old command retrieves a segment that has previously been saved either in the user's working directory or another directory to which the user has access. If the retrieval is successful, the contents of the temporary buffer are replaced by the contents of the segment specified.

Usage

```
     old {path}
```

where path is the absolute or relative pathname of a segment; if it is not supplied, the system requests it.

Example

```
!    old eval.fortran
     r


!    old >udd>design>Smith>summary.basic
     r


!    old >udd>student_lib>sort.fortran
     old:  segment not found  ">udd>student_lib>sort.fortran"
     r
```

Name: print_text, pt

The print_text command prints one or more lines of a segment. If no segment is specified, the temporary text is printed.

Usage

```
     print_text {path} {-control_arg} {first} {last}
```

where:

1.	path	is the pathname of the segment to be printed.
2.	-control_args	is one or more of the following arguments.
	-pathname path, -pn path	specifies the segment to print. This argument is only needed if the segment name begins with a digit.
	-no_header, -nhe	suppreses header line
3.	first_line	is the line number of the line at which to begin printing
4.	last_line	is the line number of the line at which to stop printing

Example

```
!    print_text

     "min.basic"  11/07/76  1215.2  mst Fri

     105 if x < y  then 140
     120 if x < z than 150
     130 let z = x + y
     140 print x, y, z

     r
```

```
!     print_text 105
      105 if x < y then 140
      r

!     print_text 120 140
      120 if x < z then 150
      130 let z = x + y
      140 print x, y, z
      r
```

Notes:

If the first_line is not specified, a short identifying header is printed preceding the printing of the segment. This header is suppressed whenever the first_line is specified.

The first_line and last_line numbers do not have to appear in the segment. The command will print all lines that fall within the range of the two numbers.

If only the first line is specified, then just that line will be printed.

Name: ready_off, rdf

The ready_off command suppresses the system promot character "r".

Usage

```
     ready_off
```

Example

```
!    print_text 10 20
     10 print "totals", "average"
     12 input x
     14 if x < 0 then 35
     r

!    ready_off
!    print_text 10 20
     10 print "totals", "average"
     12 input x
     14 if x < 0 then 35
!    save
```

Name: ready_on, rdn

The ready_on command restores the system prompt character "r".

Usage

```
     ready_on
```

Example

```
!    ready_on
     r
```

Name: resequence, rsq

The resequence command rerumbers specified lines in the temporary buffer or a saved file, beginning with a given line number and adding a given increment to derive subsequent rumbers.

NOTE: This command does special editing for BASIC source text. It changes all line references to correspond to the new line numbers. In all other cases, only the line numbers at the beginning of the line are changed. A BASIC source text is recognized by its name. If the name ends with ".basic" then the special editing will be done.

Usage

```
   resequence {path} {-control_arg} {new_number} {increment}
```

where:

1.	path	is the pathname of the segment to be resequenced. If the pathname is not given the temporary buffer is used.
2.	-control_arg	is -pathname path or -pn path where path is a pathname of the segment to be resequenced. This argument need only be used if the path begins with a digit.
3.	new_number	is the first new line number to be assigned (100 by default).
3.	increment	is the increment used to derive subsequent line numbers (10 by default).

Example

```
!     old prog.basic
      r
```

```
!     print_text -nhe
!     210 if m>n then 260
!     220 next i
!     230 if n<>m then 260
!     240 print "ok"
!     250 stop
!     260 go to 400
!     resequence
      r

!     print_text -nhe
      100 if m>n then 150
      110 next i
      120 if n<>m then 150
      130 print "ok"
      140 stop
      150 go to 400
      r


!     resequence prog.basic  400  20
      r
```

Name: run

The run command executes a BASIC or FORTRAN program. After execution it closes all input/output files and frees common blocks.

If pathname is not given, the run command compiles the source code in the temporary buffer and executes it. In order for the command to work, the default name (the name used with the last old, new or save command) must have a language suffix.

If a pathname is given, the run command executes that object segment.

Usage

```
run {pathname}
```

where:

1. pathname is the pathname of an object segment

Example

```
!     old test.basic
      r


!     run
      (program execution)
      r


!     run std            /object segment "std" in working directory
      (program execution)
      r


!     run                /temporary buffer is not changed by run
      (program execution of test.basic)
      r
```

Name: save

The save command causes the contents of the temporary buffer to be saved either in the user's working directory or in a specified directory. If no argument is supplied, it is saved under the pathname (given with the last new, old, or save request). If path is supplied, the segment is saved under the name given and in the directory given.

Usage

```
     save {path}
```

where path is the absolute or relative pathname under which the segment is to be saved.

Example

```
!    save >udd>Smith>newprog.fortran
     r


!    old scores.basic
     r


!    10 data 87,93,78,40
!    save            /the name from the old command is used
     r
```

Name: set_acl, sa

The set_acl command manipulates the access control lists (ACLs) of segments and directories. See "Access Control" in Section III.

Usage

```
     set_acl   path   mode1   {User_id1   ...   moden   User_idn}
{-control_args}
```

where:

1. path	is the pathname of a segment or directory. If it is -wd or -working_directory, the working directory is assumed. The star convention can be used.
2. mode*i*	is a valid access mode. For segments, any or all of the letters rew; for directories, any or all of the letters sma with the requirement that if "m" is present, "s" must also be present. Use null, "n" or "" to specify null access.
3. User_id*i*	is an access control name that must be of the form Person_id.Project_id.tag. All ACL entries with matching names receive the mode mode*i*. (For a descriptior of the matching strategy, see "Notes" below.) If no match is found and all three components are present, an entry is added to the ACL. If the last mode*i* has no User_id following it, the user's Person_id and current Project_id are assumed.
4. control_args	may be either of the following control arguments:
-directory, -dr	specifies that only directories are affected.
-segment, -sm	specifies that only segments are affected. This is the default.

Either control argument is used to resolve an ambiguous choice between segments and directories that occurs only when model is null and the star convention is used in path.

Notes

The arguments are processed from left to right. Therefore, the effect of a particular pair of arguments can be changed by a later pair of arguments.

The user needs modify permission on the containing directory.

The strategy for matching an access control name argument is defined by three rules:

1. A literal component, including "*", matches only a component of the same name.
2. A missing component not delimited by a period is treated the same as a literal "*" (e.g., "*.Multics" is treated as "*.Multics.*"). Missing components on the left must be delimited by periods.
3. A missing component delimited by a period matches any component.

Some examples of User_ids and which ACL entries they match are:

..*	matches only the literal ACL entry "*.*.*".
Multics	matches only the ACL entry "Multics.*.*". (The absence of a leading period makes Multics the first component.)
JRSmith..	matches any ACL entry with a first component of JRSmith.
..	matches any ACL entry.
.	matches any ACL entry with a last component of *.

""   (null string) matches any ACL entry ending in ".*.*".

Examples

```
     set_acl *.pl1 rew *
```

adds to the ACL of every segment in the working directory that has a two-component name with a seconc component of pl1 an entry with mode rew to *.*.* (everyone) if that entry does not exist; otherwise it changes the mode of the *.*.* entry to rew.

```
     sa -wd sm Jones.Faculty
```

adds to the ACL of the working directory an entry with mode sm for Jones.Faculty.* if that entry does not exist; otherwise it changes the mode of the Jones.Faculty.* entry to sm.

```
     sa alpha.basic rew .Faculty. r Jones.Faculty.
```

changes the mode of every entry on the ACL of alpha.basic with a middle component of Faculty to rew, then changes the mode of every entry that starts with Jones.Faculty to r.

Name: set_tty, stty

The set_tty command is used to modify the terminal type associated with the user's terminal and/or the modes associated with terminal I/O. The type as specified by this command is used for determining character conversion and delay timings; it has no effect on communications line control. Most users who need to use this command will need to specify only the terminal type. For special problems, see Section V in the MPM Reference Guide for a description of Multics Input/Output.

Usage

```
     set_tty {-control_args}
```

where control_args may be chosen from the following control arguments:

-terminal_type XX, -ttp XX — causes the user's terminal type to be set to device type XX, where XX can be any one of the following:

1050	device similar to IBM Model 1050
2741	device similar to IBM Model 2741, EBCDIC codes
CORR2741, corr2741	device similar to IBM Model 2741 with correspondence keyboard and 015 typeball
TTY37, tty37	device similar to Teleytpe Model 37
TTY33, tty33	device similar to Teletype Model 33 or 35
TTY38, tty38	device similar to Teletype Model 38
TN300, tn300	device similar to GE TermiNet 300 or 1220
ARDS, ards	device similar to Adage, Inc. Advanced Remote Display Station (ARDS)
ASCII, ascii	device similar to a Computer Devices Inc. (CDI) Model 1030 or Texas Instruments (TI) Model 725, or a device with an unrecognized answerback,

or a device without an answerback (these devices are collectively termed "ASCII" devices)

The default modes for the new terminal type are turned on.

-modes XX	sets the modes for terminal I/O according to XX, which is a string of mode names separated by commas, each one optionally preceded by "^" to turn the specified mode off. For a list of valid mode names, see the description of the tty_ I/O module in the MPM Subroutines. Modes not specified in XX are left unchanged. See "Notes" below.
-reset	turns off all modes that are not set in the default modes string for the current terminal type.
-tabs	specifies that the device has software-settable tabs, and that the tabs are to be set. This control argument currently has effect only for GE TermiNet 300-like devices.
-print	causes the terminal type and modes to be printed on the terminal. If any other control arguments are specified, the type and modes printed reflect the result of the command.

Notes

Invoking the set_tty command causes the system to perform the following steps in the specified order:

1. If the -terminal_type control argument is specified, set the specified device type and turn on the default modes for that type.

2. If the -reset control argument is specified, turn off all modes that are not set in the default modes string for the current terminal type.

3. If the -modes control argument is specified, turn on or off those modes explicitly specified.

4. If the -tabs control argument is specified, and the terminal has settable tabs, set the tabs.

5. If the -print control argument is specified, print the type and modes on the terminal.