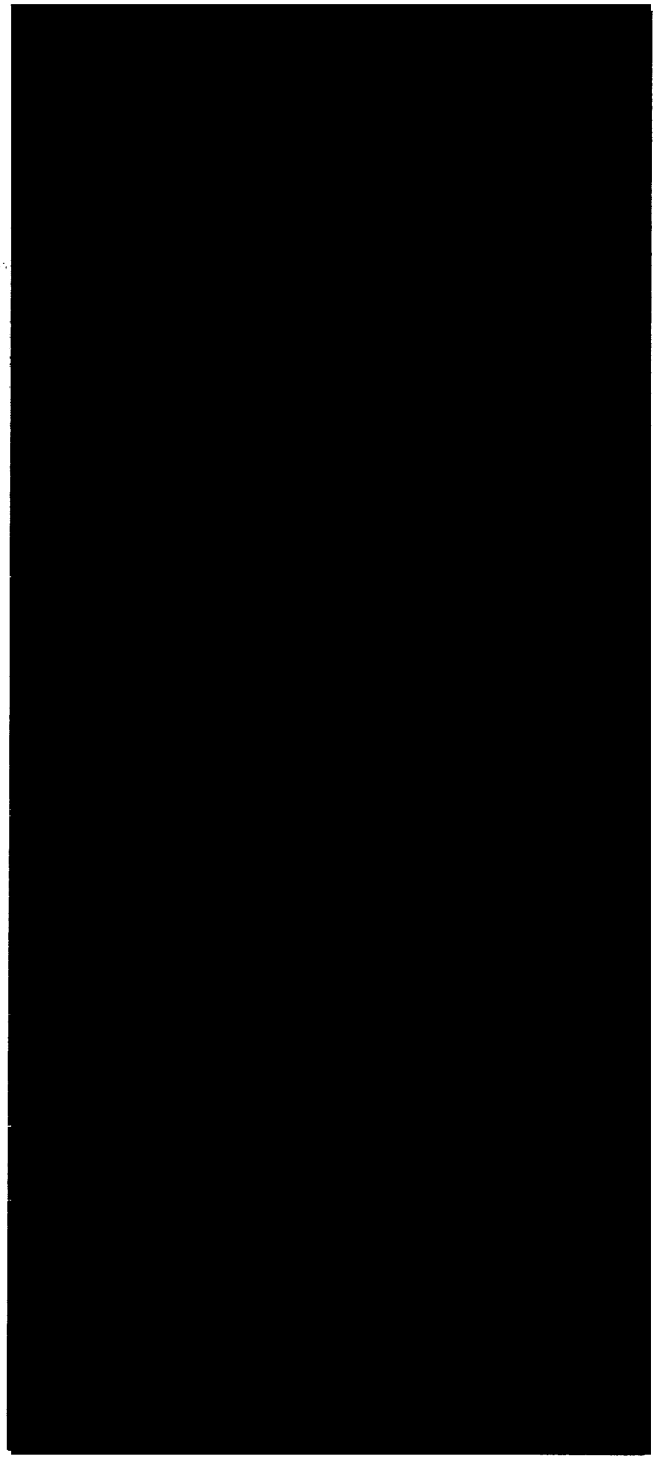
Honeywell

MULTICS DFAST SUBSYSTEM USERS' GUIDE

SERIES 60 (LEVEL 68)

SOFTWARE



Honeywell

MULTICS DFAST SUBSYSTEM USERS' GUIDE

SERIES 60 (LEVEL 68)

SUBJECT:

Description of the General Characteristics of the Multics DFAST Subsystem and the Multics DFAST Command Language

SOFTWARE SUPPORTED:

Multics Software Release 3.1

DATE:

March 1976

ORDER NUMBER:

AT59, Rev. 0

PREFACE

This document describes DFAST, a time-sharing facility supporting BASIC and FORTRAN program development. DFAST operates as a subsystem under Multics. Its command language and repertoire are based on the Dartmouth Time-Sharing System (DTSS) with additions for compatibility with the Multics storage system, access control features, and input/output facilities.

The manual presupposes no knowledge of the Multics system. BASIC programmers using DFAST are referred to the Multics BASIC manual (Order No. AM82) and to Appendix C of this document, which outlines differences implemented for DFAST BASIC. FORTRAN programmers are referred to the Multics DFAST/FAST FORTRAN Reference Manual (Order No. AT58.)

File No.: 1L13

AT59

CONTENTS

ILLUSTRATIONS

SECTION I

INTRODUCTION

DFAST is an easy-to-use time-sharing facility designed primarily for creating and running BASIC and FORTRAN programs. A simple command language is used to create and edit text files, to compile and run programs, and to select a variety of options.

The files and programs of DFAST are part of the Multics system environment in which DFAST operates. The DFAST command repertoire and language conventions are based on the Dartmouth Time-Sharing System (DTSS) with extensions for compatibility with Multics. In addition, a small set of Multics commands have been added to the DFAST language to provide user control of Multics file access and input/output mechanisms. No knowledge of Multics is required to use DFAST.

This manual is intended to permit the programmer to use DFAST immediately. The introductory information in this section and the sample session in Section II provide enough information to begin using DFAST. A complete overview of DFAST is given in Section III. Section IV gives detailed descriptions of each of the DFAST commands. Section V describes text editing facilities.

DFAST FEATURES

The user interacts with DFAST in an online session, issuing commands and awaiting response. The major activity during a user-DFAST dialogue is centered on creating and manipulating the current file (a unit consisting of all input entered by the user during the session), a file retrieved from the Multics storage system (permanent online storage), or an object program produced by one of the DFAST compilers.

DFAST BASIC is similar to the original Dartmouth version, differing from standard Multics BASIC as described in Appendix B. The system name "basic" selects BASIC with single-precision arithmetic. The system name "dbasic" selects BASIC with double-precision arithmetic. Use of both precisions is not allowed in the same program run and files produced by one version are not compatible with those produced by the other (basic uses one word to store numbers, dbasic two).

The version of FORTRAN used on DFAST is a superset of ANSI FORTRAN (1966). A number of time-sharing oriented features have been added and the use of expressions in language constructs generally expanded. The DFAST FORTRAN language is described in the Multics DFAST/FAST FORTRAN Reference Manual (Order No. AT58).

Automatic editing and sorting of line-numbered input is provided. In addition, a set of edit requests can be used to modify existing text lines or reorganize and renumber an entire file.

File handling facilities support file creation, deletion, modification, and renaming. A user can access any file in the Multics system to which he has the appropriate access privileges. This means that the user can use programs that belong to other users or programs from system libraries.

DFAST maintains a variety of online information available to a user on request. This includes brief descriptions of DFAST commands, information on the current state of the DFAST subsystem, and file-related information.

USING DFAST

To begin a DFAST session, the user logs in to the Multics system. After the Multics initial message has been typed, DFAST issues a ready message in the form:

```
ready  0900
```

This message is printed throughout the session to inform the user that DFAST has completed a specified task and is again ready to accept user input. The time of day is printed with each ready message.

User input can be a command or text. Input text usually must begin with a line number. The build command, described in Section IV, can be used to enter nonnumbered lines.

At the end of the session, a user must log out.

Logging In

A DFAST user must be registered under a project associated with DFAST. He will be assigned a unique identification (called a Person_id) and a password, both of which must be entered precisely as assigned whenever he logs in. If the user's Person_id is JBrown, he cannot log in if he types Jbrown or J Brown.

The password is entered either superimposed on a string of cover-up characters or with printing suppressed, to ensure confidentiality.

A sample login, including the messages printed by Multics and DFAST is shown below. Prior to this interchange, the user must dial the appropriate telephone number to establish a connection with Multics. The exclamation point (!) is used here and throughout this document to denote text typed by the user; this should not actually be typed by a user.

```
     Multics MRX.X: Multics Service, PCO,Phoenix,AZ.
     Load = 26.0 out of 100.0 units: users = 26

!    login JBrown
     Password:
!
     You are protected from preemption until 0829.
     JBrown Multics logged in 01/27/76  0729.2 mst Tue from ASCII terminal "none".
     Last login 01/26/76  1230.0 mst Mon from ASCII terminal "none".
     ready  0729
```

The ready message indicates that the user, JBrown, is successfully logged in and that DFAST is awaiting input. Additional messages may be printed to provide general information such as the addition of features, scheduled shutdowns, and so on. Errors during logging in are described by messages such as:

```
Login incorrect
Please try again or type "help" for instructions.
```

Typing Conventions

User-typed lines can contain commands or input text but not both. Usually, the user types one command or line of text per physical line, terminating a line with the appropriate carriage-control character. After a command line, DFAST issues a ready message and spaces down one line.

Typing errors can be corrected using the special symbols # and @. The number sign (#) indicates that the character immediately preceding it should be deleted. To delete a character five positions back, five #'s must be typed, deleting all characters back to that point. (An exception to this is when blanks or tabs are intervening; one # deletes all white space.) Some examples of the use of # are given below. In each example, an exclamation point precedes the line typed by the user and the line beneath it shows what the final input is.

```
!     new newfa#ile.basic
      new newfile.basic

!     new new###oldfile.fortran
      new oldfile.fortran

!     mew new#######new newfile
      new newfile
```

The commercial at sign (@) deletes an entire line.

```
!     new new@new anotherfile
      new anotherfile
```

Quit Signal

The user can interrupt DFAST during command or program execution by depressing the ATTN, INTERRUPT, BRK, or QUIT key on the terminal. DFAST returns to command level and issues a new ready message.

Case Conventions

Input from twocase terminals is stored as entered by the user. Input from onecase terminals is stored as all lowercase. If a user wishes to enter a capital letter, the input must be preceded by a backslash (\). For the special characters # and @ to be stored as characters (suppressing their erase functions), they must be preceded by a backslash on all terminals. Nonprinting characters are input with a backslash followed by their octal representation.

Output conventions can be controlled by the user with the DFAST commands onecase and twocase. At login, the output mode is twocase. Characters are printed exactly as stored. Thus, a capital Z is printed as \Z on a onecase terminal and simply as Z on a twocase. A lowercase z is printed as Z on a onecase terminal and as z on a twocase. A nonprinting character is typed as a backslash followed by the octal representation.

In onecase output mode, both lowercase and uppercase letters are printed as uppercase on all terminals and nonprinting characters are suppressed.

Logging Out

When a user has completed a session, he must log out. To log out and disconnect the terminal, he can issue either of the commands bye or goodbye (some terminals require that the user manually disconnect the acoustic coupler). The hello command logs the user out, maintaining the connection for the next user.

Error Handling

When a user makes an error in a command line, DFAST issues a descriptive error message of the form:

```
command_name:  message
```

Several commands may invoke the same error message. For example, "unknown argument" can be issued by most commands. When a DFAST error occurs, the user is issued a new ready message and can retype the command or input line that caused it. If the user has a question about an error, he can obtain an online description of the command that caused it using the explain command (using "explain topics" he can determine if there is an online description of a general topic such as file access).

The sample session excerpted below shows an error message printed by DFAST.

```
!    compile
     compile:  current segment must be saved "test.basic"
     ready  0910

!    save
     ready  0910

!    compile
     ready  0911
```

Here, the user had to save the current file before compilation could be successfully performed (the compile command causes the source text in the current file to be replaced by the object code generated).

SECTION II

SAMPLE SESSION

The following session shows the application of DFAST commands to the compilation and execution of a BASIC program. User typing is indicated by the exclamation point character (!). Comments are to the right and preceded by the slash character (/). Full descriptions of the commands used in the sample session are given in Section IV.

The user begins the session by dialing into the Multics system and receives a response before logging in.

```
     Multics MRX.X: Multics Service, PCO,Phoenix,AZ.
     Load = 11.0 out of 80.0 units: users = 11
!    login Smith
     Password:
!
     You are protected from preemption until 0820.
     Smith Multics logged in 01/27/76  0720.2 mst Tue from ASCII terminal "none".
     Last login 01/26/76  1230.0 mst Mon from ASCII terminal "none".
     ready  0720
```

To begin entering input, the user issues the DFAST command, new, and supplies a name for the current file.

```
!    new parens.basic
     ready  0720
```

Any name that adheres to the naming conventions given in Section III can be assigned. Here, the user is entering a BASIC source program, indicated by the second component of the name. Input to the file begins now. This is a program intended to check for matching parentheses in any input string.

```
!    100 input l$
!    110 let n = 0
!    120 let m = 0
!    130 for i = 1 to len (l$)
!    140 let b$ = seg$ (l, i, i)
!    150 if b$ <> "(" then 200
!    160 let n = n +1
!    200 if b <> ")" then 220
!    210 let m = m + 1
!    215 if m>n then 260
!    220 next i
!    230 if n=@230 if n <> m then 260
!    240 print "parens match"
```

```
!     250 stop
!     260 print "parens don't match, try again"
!     270 goto 100
!     save
      ready  0722
```

The user here has saved his current file permanently before running it under the name assigned with the new command. The current file can be saved under another name by supplying a name as an argument to the save command.

```
!     run
      String expression required in 140              /BASIC error messages
      Mixed string and numeric expression in 200
      No end statement as of 270
      run:  error in compilation "parens"            /DFAST error message
      ready  0722
```

The user has issued a run command that ordinarily compiles a program and then executes it immediately. This time, he has received error messages from BASIC as well as an error message from DFAST. The compiler used by default was BASIC because it was the current system at this time. If the current system had been FORTRAN, the user could have issued the command "run basic," which would have reset the current system. To ascertain the current system as well as the name of the current file, the user issues a tty command.

```
!     tty
      name = parens.basic,  system = basic,  user = Smith.Design,  line = tty020
      ready  0723
```

In order to successfully compile his program, the user must correct the source code. This can be done in DFAST simply by typing new lines with the same numbers to replace errors. To delete a line, the user can simply type a line number immediately followed by a carriage return.

```
!     140 let b$ = seg$ (1$, i, i)
!     200 if b$ <> ")" then 220
!     280 end
```

These corrections are added to the current file. To store them on the saved copy, the user must overwrite the saved version with the contents of the current file.

```
!     replace parens.basic
      ready  0724
```

To obtain a listing of the source code, the user can issue a list or listnh (abbreviated to lisn) command. Here, lisn, which suppresses header information such as name and date, has been selected.

```
!     lisn
      100 input l$
      110 let n = 0
      120 let m = 0
      130 for i = 1 to len (l$)
      140 let b$ = seg$ (l, i, i)
      150 if b$ <> "(" then 200
      160 let n = n +1
      200 if b$ <> ")" then 220
      210 let m = m + 1
      215 if m>n then 260
      220 next i
      230 if n <> m then 260
      240 print "parens match"
      250 stop
      260 print "parens don't match, try again"
      270 goto 100
      280 end
      ready  0727
```

Now, the user wishes to recompile the program. Instead of the run command, the user decides to use the compile command, which compiles and, if successful, returns the object code as the current file. It can then be saved for subsequent execution. In the sample program, the user types in a string of characters when the ? character is printed by the program.

```
!     compile
      ready  0728


!     save
      ready  0728


!     run
      ?  ((()))
      parens don't match, try again
      ?  ))((
      parens don't match, try again
      ?  (12(20(abcd)e)f)
      parens match
      ready  0730
```

To end the session, the user logs out by issuing the bye command.

```
!     bye
      Smith Multics logged out 01/27/76 0731.3 mst Tue
      CPU usage 5 sec, memory usage 16.5 units.
```

If, in the future, the user wishes to change or add on to his program, he can retrieve the source file parens.basic by issuing the command "old parens.basic". The original source program then becomes the current file and can be changed by typing replacement lines or using the edit command to invoke functions such as deletion, insertion, and resequencing.

SECTION III

COMMAND LANGUAGE OVERVIEW

The DFAST command language is based on a set of commands that describe general functions to be performed. Many of these commands require arguments to particularize the function. For example, the save command, which stores a file in the Multics storage system, requires a name under which to store it. The user need not always supply such an argument since most DFAST commands operate on default assumptions based on the current state of the command environment. In the case of save, when the user supplies no argument, the current file is stored under its current name.

DFAST LANGUAGE CONVENTIONS

User-supplied arguments to commands must adhere to appropriate naming conventions; that is, file names must be constructed according to the rules given under "File Naming Conventions" below, line numbers to conventions given under "Input Lines" below, and so on.

File Naming Conventions

A file 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 hyphen (-), underscore (_), and period (.). A period has a special effect, dividing a user construct into separate components to be interpreted by DFAST. 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 file is a FORTRAN source program.

By convention, an asterisk (*) can be used to represent any component when a file name argument is given in a command such as catalog, which searches the storage system. Called the star convention, the asterisk in this context means "any text." Thus, "*.fortran" would indicate all files with a two-component name whose second component is fortran. Two asterisks can be used together to represent any number of components (including none). For example, "test.**" would match any of the following: test, test.basic, test.fortran, test.new.basic.

A DFAST file is equivalent to a Multics segment, the basic unit of the Multics storage system. On Multics, directory segments are maintained for use in locating other segments (including other directory segments). Directories maintain a tree-structured hierarchy that permits any segment to be referenced by the series of directories leading from the root of the tree to the target segment (or DFAST file.) Each user is assigned his own directory (home directory) at login to which all his storage system requests refer by default (that is, when he does not specify some other directory explicitly). When a file becomes part of the storage system, its name identifies its position in this tree-structured hierarchy. Figure 3-1 shows a portion of this structure in a very simplified form.

Based on the sample hierarchy of Figure 3-1, if user TSmith wants to use user BJones' file called test.basic, he would provide a name indicating the list of directories leading to BJones (Jones' home directory) and terminating with the desired file. This string of names is called a pathname. By convention, individual names in a pathname are separated by the greater-than character (>) and the root directory need not be specified. Thus, the full pathname for Jones' file named test.basic would be:

```
>udd>ProjA>BJones>test.basic
```

Notice that this notation permits users TSmith and BJones to have files with the same name. Smith's test.basic file would be specified by Jones as:

```
>udd>ProjA>TSmith>test.basic
```

By convention, DFAST permits a user to specify files in his home directory by name alone. Thus, if a DFAST user means to specify his own copy of test.basic he need type only "test.basic". To specify another user's test.basic file, however, he must still supply a full pathname.

Command Lines

A line containing a single command can begin at any horizontal position. When arguments are supplied, at least one blank must separate them from the command. Arguments are separated from each other by blanks, and the entire line is terminated by a newline character (ASCII code 012).

More than one command can appear on a line if a separator is used. By convention in DFAST, if the first character typed is not alphanumeric, it will be interpreted as a separator, as in:

```
/rename newfile/save/compile/
```

This is equivalent to the sequence:

```
rename newfile
save
compile
```

When a number of commands appear on a line, DFAST executes all of them before issuing a single ready message.

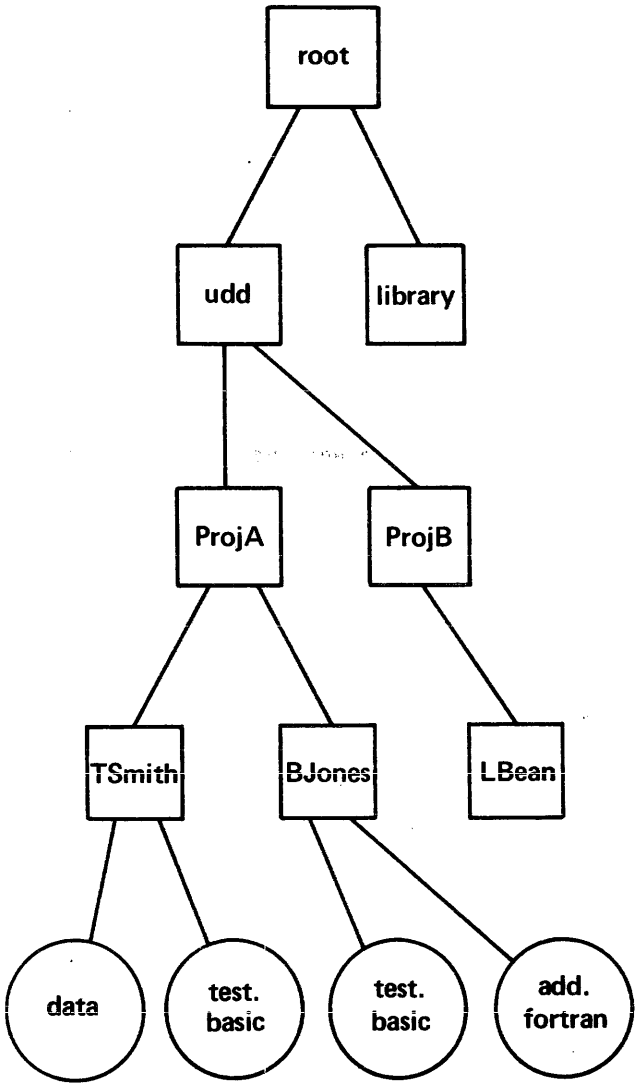


Figure 3-1. Storage System Hierarchy

Input Lines

Any line that begins with a number is interpreted as a line of input text, except within the context of an executing user program. Preceding blanks are ignored. All of the following lines will be entered into the current file.

```
100 enter
110 new
5   text
    7 here
```

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.

Text without line numbers can be entered using the build command.

COMMAND ENVIRONMENT

The effect of a particular DFAST command can vary at different executions, depending on the current state of the environment. For example, if a user has just compiled a FORTRAN program, the current system becomes FORTRAN and must be changed to BASIC before a BASIC program can be compiled. There are four elements of the command environment that can affect the use of commands. These are:

```
current file
alter file
current name
current system
```

At login, these have the following values:

current file	empty
alter file	empty
current name	"no name"
current system	basic

Subsequent user-DFAST interaction changes these values. When a command uses one of these as a default value, the most recent change is in effect.

Current and Alter Files

The file that a user creates in a DFAST session is called the current file. All information entered into the current file is first temporarily stored in a buffer called the alter file. To begin a new current file, the user issues the command, new, followed by line-numbered input. The input lines are stored in the alter file. When a command is issued that acts on the current file, the alter file is sorted, then merged with the previous contents of the current file. When the alter file is sorted, lines are put in ascending numerical sequence. When duplicate line numbers occur, the last line entered is retained.

The sample below shows the user-DFAST dialogue on the left and the corresponding contents of the current and alter files to the right.

```
!    new newfile
!    100 new text                alter file          current file
!    110 for new file            90  This is
!    90  this isn't              100 new text        (empty)
!    90  This is                 110 for new file

!    save
     ready  0610                 alter file          current file
                                 (empty)             90  This is
                                                     100 new text
                                                     110 for new file

                                 alter file          current file
!    80  now                     80  now             90  This is
!    120 sample                  90  here is         100 new text
!    90  here is                 120 sample          110 for new file

                                 alter file          current file
!    replace newfile             (empty)             80  now
     ready  0610                                     90  here is
                                                     100 new text
                                                     110 for new file
                                                     120 sample
```

The contents of the current file can also be changed by other methods. For example, a previously saved file can be retrieved using the command, old. When the current file is a source program to be compiled, the current file after compilation is changed to the resultant object program. In these cases, as with build, the alter file is not used.

Current Name

The current name of a file is initially "no name"; that is, the character string "no name" is used on listings where the name would normally appear. The current name can be explicitly assigned by providing a name argument with the command, new, or by executing a rename command. When the command, old, is issued, the current name is automatically changed to the name of the retrieved file. The current name is also changed as a byproduct of a successful compilation using the compile command, which returns object code as the current file. In this case, if the source program name has a language suffix of "basic" or "fortran," then the current name is changed to the name preceding the suffix (for example, "test.basic" becomes "test"). If the source program does not have a language suffix, the current name is changed to "object."

Current System

The current system is the compiler (basic, dbasic, or fortran) that is used by default in the compile and run commands. It is also used in connection with the resequencing facility of the edit command (see Section V). At logging in time, the current system is basic. It can be explicitly reset by a compile, system, or run command. It is automatically changed by the old, rename, and new commands as follows. If the name referenced by old has a language suffix or is an object program, the system name is changed to the corresponding compiler. If rename or new has an argument with a language suffix, system name is set to the appropriate compiler.

Each file stored in the Multics storage system has a set of access rights associated with it. By default, a user has complete access to all files in his home directory, and access is denied to any other user. The user has control of these rights and can specify both those users who can have access to a particular file and the type of access. For example, a user may specify that anyone can have access to read a particular file but that only he himself can have access to write on it.

Access Control List

The access rights for each file are described in its access control list (ACL). An ACL contains the identification of users permitted (or specifically denied) access to the file plus a description of the type of access allowed.

The user identification in the ACL consists of a three-component name: Person_id, Project_id, and an instance tag, separated by periods. The Person_id is as described under "Logging In" in Section I. The Project_id is the identification of the user's project, registered by a Multics system administrator. Multics assigns the instance tag when the user logs in. Whenever anyone tries to access a file on the Multics system, his three-component name must match one of the entries on the ACL of that particular file; if not, he has no access to that file.

Access Modes

The type of access allowed is defined by access modes: four modes for files and four modes for directories.

Access modes for files are:

read	(r)	data in the file can be read
write	(w)	data in the file can be written (modified)
execute	(e)	an executing process can transfer to and execute instructions in this file
null	(n)	access to the file is denied

Access modes for directories are:

status	(s)	the attributes of files contained in the directory can be obtained
modify	(m)	the attributes of existing files contained in the directory can be modified (changed or deleted)
append	(a)	new files can be created in the directory
null	(n)	access to the directory is denied

Generally, combinations of access modes are assigned to files and directories. Typical access mode combinations are:

Files	Directories
r	s
w	sm
re	sa
rw	sma
rew	null
null	

The user can specify access mode assignments for files only, although he can list the access on directories. Once specified, the access is not "frozen"; the user may change it at will by specifying different modes, persons, or projects as arguments to the set_acl command, described below.

Setting Access

The command the user invokes to set the ACL, set_acl (described in detail in Section IV, "Command Descriptions"), either adds an entry to the ACL or modifies an existing entry. The set_acl command, which may be abbreviated sa, has the general format:

```
sa  file_name  mode(s)  User_id
```

For example, Tom Smith has text in file xsolve of his directory that Jane Doe wants to use. To give her access so she can read the file, he types:

```
! sa xsolve r JDoe.*.*
```

If he instead decides that his file should not be available to Jane and wants to make sure she cannot read it, he types:

```
! sa xsolve null JDoe.*.*
```

The asterisk following Jane's Person_id (JDoe) in the above command lines means that the requested access applies to Jane no matter what project she may be on, no matter what instance tag may be associated with her work. For example, the User_id Tom gave, JDoe.*.*, matches:

```
JDoe.ProjB.*
JDoe.ProjA.*
JDoe.ANYTHING.*
```

When the user wants to denote any Person_id, he types an asterisk for the first component; any Project_id, an asterisk for the second component; and any instance tag, an asterisk for the third component. (It is best to use an asterisk for the third component since the user generally does not know the instance tag.) Thus, a user identification of *.*.* specifies any user.

Listing Access

To check the ACL of a file, the user invokes the command that lists the ACL, list_acl (described in detail in Section IV, "Command Descriptions"). The list_acl command, which may be abbreviated la, has the general format:

```
la  file_name
```

As explained earlier, any file_name that does not begin with the greater-than symbol is assumed to be in the user's home directory. Thus, if Tom Smith wants to list the ACL of xsolve, he types:

```
! la xsolve
   rw       TSmith.ProjA.*
   r        JDoe.*.*
   rw       *.SysDaemon.*
   r        *.ProjA.*
```

The third entry in the example, *.SysDaemon.*, identifies various Multics system processes that control such things as offline printing and making copies of files or "backup" tapes. Appropriate ACL entries are placed on every file the user creates so these system processes will have the necessary access to perform the various backup, metering, and input/output functions.

If Tom is interested in checking the access he has given only Jane on xsolve, he types:

```
! la xsolve JDoe
   r        JDoe.*.*
```

or to check the access rights of only ProjA, he types:

```
! la xsolve .ProjA
   rw       TSmith.ProjA.*
   r        *.ProjA.*
```

Notice that when specifying the user identifications, periods must be used to show "missing" components to the left of a specified component; however, it is not necessary to include periods for "missing" components on the right.

Deleting Access

A third access control command, delete_acl, allows the user to delete ACL entries. This command, which may be abbreviated da, has the same general format and rules as the list_acl command. (See Section IV, "Command Descriptions," for a detailed description of delete_acl.)

For example, if Tom Smith has changed file beta, he might want to also change its ACL. First, he lists the ACL entries to see who currently has access to beta:

```
! la beta
   rw       TSmith.ProjA.*
   re       Gray.Merlin.*
   rw       Butler.Merlin.*
   rw       Jones.*.*
   re       JDoe.*.*
   rw       *.SysDaemon.*
   r        *.*.*
```

Tom decides that he no longer wants user Jones, anyone on the Merlin project, or the entire user community (represented by *.*.*) to have access to beta. Therefore, he invokes the delete_acl command in the following manner:

```
! da beta Jones *.*.* .Merlin
```

If Tom now again invokes list_acl, he will see that the requested change has already taken place.

```
! la beta
   rw       TSmith.ProjA.*
   re       JDoe.*.*
   rw       *.SysDaemon.*
```

Changes in access rights occur instantaneously. If Jane has access to a file of Tom's, and he changes the access while she is using the file, DFAST prints out a message telling her that she has incorrect access to the file and returns her to command level.

DFAST COMMAND REPERTOIRE

The complete repertoire of DFAST commands is given below, organized in terms of general function. A detailed description of each of these commands is provided in Section IV.

Logging In/Logging Out

enter, enterp	connects anonymous user to the Multics system.
goodbye, bye	terminates a user session and disconnects the terminal.
hello	terminates a user session but leaves the terminal connected for subsequent user.
login	connects registered user to the Multics system; used at dialup or after a hello command.

File Creation and Edit

append	appends unsorted contents of alter file to current file.
build	initiates non-line-numbered mode of input.
edit	performs text-editing requests.
ignore	discards contents of the alter file.
list, listnh	lists all or portions of the current and/or alter files (listnh suppresses header information).
new	initiates a new current file, deletes both the current and alter files, and changes the current name.
scratch	deletes both the current and alter files.
sort	sorts the current file into ascending line-number sequence.

File Storage and Retrieval

catalog	requests information about files stored in specified directories.
old	retrieves a previously saved file and makes it the current file.
replace	replaces the contents of a previously saved file with the contents of the current file.
save	creates a new file that contains a copy of the contents of the current segment.
unsave	deletes a stored file.

Access Control

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

Command Environment

rename	renames the current file.
system	resets the current system (compiler).

Information

bill	prints accounting information.
explain	prints online description of specified topic.
help	at login, prints login information; otherwise like explain.
length	prints the number of words in the current file.
tty	prints current command environment values.
users	lists users currently logged in.

Input/Output

brief	establishes brief output mode for printing.
dprint	queues a file for printing on the high-speed line printer.
nbrief	terminates brief output mode.
onecase	establishes single-case input/output mode.
set_tty	establishes new terminal type.
twocase	establishes two-case output mode.

Programming Facilities

compile	compiles the source program in the current file.
run	compiles, if necessary, and runs a user program.

SECTION IV

COMMAND DESCRIPTIONS

This section contains, in alphabetical order, a description of each of the DFAST 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: append, app

Here, the append command can be invoked by typing either "append" or "app."

USAGE

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

compile {system_name}

Here, system_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-DFAST 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: append, app

The append command appends all information currently contained in the alter file to the current file; that is, information is added at the end of the file instead of being merged into the appropriate line-number sequence.

Prior to execution of an append command, the alter file contains all information entered since the last command that caused a merge of the alter and current files such as new, old, or replace. After execution, the alter file is empty.

Usage

```
append
```

Example

```
!    new new_file
     ready  1301

!    100 this is old
!    110 text
!    save
     ready  1301

!    100 this is new
!    110 text
!    lisn current
     100 this is old
     110 text
     ready  1301

!    lisn alter
     100 this is new
     110 text
     ready  1302

!    append
     ready  1302

!    lisn current
     100 this is old
     110 text
     100 this is new
     110 text
     ready  1302
```

Name: bill, bil

The bill command prints a record of charges for computer usage by the user. The output gives the date covered by the report, the total charges, and a breakdown of charges.

Usage

```
bill
```

Example

```
!    bill

     Smith.Design  Report from 10/22/75  1935.1 to 10/30/75   1001.3

       Month-To-Date Charge: $    37.40;
       Resource Limit:       $   100.00;

          Interactive Usage: $   36.66;  13 logins, 0 crashes.

               shift  $charge    $limit
                1       32.56      open
                3        4.10      open

          Absentee Usage:    none;


          IO Daemon Usage:   $    0.64;

               queue  $charge   lines/K
                3       0.74         1

     ready  1002
```

Name: brief, bri

The brief command suppresses the DFAST-issued ready message and the header preceding a printout by the list command.

Usage

```
brief
```

Example

```
!    list alter

     alter      12/2/75      1210.2    mst    Mon

     100 random text
     ready  1210

!    brief
!    list alter
     100 random text
     ready  1210
```

Name: build, bui

The build command initiates an input mode for nonnumbered lines of text that are appended directly to the current file. Any text in the alter file when build is given is merged before the new text is appended. (Notice that a DFAST command entered in this mode is simply accepted as text.) The build mode of input is terminated by typing a line consisting of a newline. When DFAST issues a ready message, the normal command environment is restored.

Usage

```
build
```

Example

```
!    new test
     ready  0925

!    100 this is
!    110 a test
!    build
!    of lines typed
!    save
!    replace
!    etcetera
!
     ready  0925

!    lisn
     100 this is
     110 a test
     of lines typed
     save
     replace
     etcetera
     ready  0926
```

Name: bye

The bye command terminates a user session and ends communication with the DFAST system.

On terminals equipped with acoustic couplers, it is necessary to hang up the telephone handset.

Usage

bye

Example

```
!    bye

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

Name: catalog, cat

When a file is saved, its name and other information about it is placed in the directory specified (by default, the user's home directory). To print information about the files in a single directory, the user can issue a catalog command. A variety of control arguments allow the user to restrict the listing to a subset of files and/or a subset of information. When no arguments are given, the command prints the name, access mode, and length for each file in the home directory in the order in which they were created. The star convention is allowed (see "File Naming Conventions" in Section III).

Usage

catalog {file_names} {-control_args}

where:

1.	file_names	are a subset of the files whose attributes are to be listed. Listing of information about these files depends on the control arguments given.
2.	control_args	may be chosen from the arguments given below and supplied in any order. The basic output format of catalog is a series of columns, each of which corresponds to an attribute of the file. 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. Files are printed in the order they occur unless the user explicitly requests a different order.
	-pathname path, -pn path	lists the contents of the directory specified by path; if this control argument is not supplied, the home directory is assumed.
	-name, -nm	prints only the names column.
	-date_time_entry_modified, -dtem	prints the date and time the file was last modified.
	-total, -tt	prints only the heading line, giving the total number of files (Multics segments) and the sum of their sizes.
	-no_header, -nhe	omits all heading lines.

catalog

Example

```
!     catalog

      Segments = 4, Lengths = 26.

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


      ready  0910

!     catalog *.basic

      Segments = 2, Lengths = 12.

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


      ready  0910

!     catalog *.basic -nm -nhe

      test.basic
      summary.basic


      ready  0911

!     catalog -tt

      Segments = 4, Lengths = 26.


      ready  0911
```

Name: compile, com

The compile command compiles the current program into object code by the BASIC or FORTRAN compiler. The resultant object program becomes the current file and can be executed immediately using the run command or can be saved for subsequent execution. The current file must be saved before compilation.

The current name is changed, as follows, with respect to the source program name. If the source program name has a language suffix (e.g., prog.basic), the current name after compilation becomes the source name with the suffix removed (e.g., prog). If no suffix was used for the source program (e.g., prog), the current name becomes "object." If errors are detected during compilation, error messages are issued by the compiler and the source program is retained as the current file.

Usage

```
compile {system_name}
```

where system_name is basic, dbasic, or fortran.

If no argument is supplied, the current system is the value assumed. For information on determining the current system, see "Current System" in Section III.

Example

```
!    rename test.basic
     ready  1100

!    compile
     compile: current segment must be saved
     ready  1100

!    save
     ready  1100

!    compile
     ready  1100

!    tty
     name = test,  system = basic,  user = Smith,  line = tty112
```

Name: delete_acl, da

The delete_acl command removes entries from the access control lists (ACLs) of files. See "Access Control" in Section III.

Usage

delete_acl {file_name} {User_ids} {-control_args}

where:

1. file_name — is the name of the file whose ACL is to be deleted. If it is omitted, only a User_id of -all or -a is allowed. 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 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.*. If no User_id is given, the user's Person_id and Project_id are assumed.

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."

Note

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 on tape, Dumper.SysDaemon.* from reloading it, and Retriever.SysDaemon.* from retrieving it.

Example

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

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

delete_acl

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

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

Name: dprint, dp

The dprint command queues specified files for printing on the line printer. The output begins with a header sheet that is identified by the requestor's User_id and, if specified, the destination. A summary sheet indicates the time of the request, the time of printing, the number of lines and pages printed, and the cost of printing.

Usage

dprint {-control_args} {file1 file2 ... filen}

where:

1.	control_args	may be chosen from the following list of control arguments and can appear anywhere in the command line:
	-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 given, the default is the requestor's Project_id. This argument can be overruled by a subsequent -destination control argument.
	-map	prints a file using only uppercase letters. See "Notes" below.
2.	filei	each filei is the name of a file to be queued for printing.

Notes

The dprint command, invoked without any arguments, prints a message telling how many requests are in the queue for printing.

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

If the -map control argument is used, an uppercase version of the user's file is created in his home directory with the name "file_name.map". After printing, it is deleted. Only one file can be printed by dprint when the -map control argument is supplied.

Example

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

causes a copy of each of the files named test.basic and test.fortran in the home directory to be printed with the header "Jones".

Name: edit, edi

The edit command invokes a specified text-editing function. The desired function is expressed as one of the keywords given under "Usage" below with arguments as required by a specified function. A detailed description of all edit functions is given in Section V, "Text Editing."

Usage

edit function

where function may be selected from one of the following:

Function	Effect
append	combines two or more files and resequences line numbers.
delete	deletes one or more lines in current file.
desequence	removes line numbers from current file.
explain	prints online description of specified edit request.
extract	selects specified lines to be retained when current file is deleted.
insert	inserts the contents of one or more files at specified locations of the current file.
join	combines two or more files without resequencing.
list	requests printout of all or a portion of the current file.
locate	requests a listing of lines containing a specified text string.
merge	merges and sorts the contents of two or more files.
move	relocates one or more lines within the current file.
prefix	inserts a given character string before existing string.
replace	substitutes new character string for existing one.
resequence	assigns a new set of line numbers to all or a portion of the current file.
sequence	assigns a new set of line numbers to an entire current file.

string	converts the current file to a random-access string file for use with BASIC and FORTRAN programs.
suffix	inserts given character string after existing one.

enter

Name: enter, e
enterp, ep

These requests are used by anonymous users to gain access to DFAST. Either one is actually a request to the answering service to create a process for the anonymous user. See also the login command.

Anonymous users who are not to supply a password use the enter (e) request. Anonymous users who are to supply a password use the enterp (ep) request.

Usage

enter {anonymous_name} Project_id {-control_args}

where:

1.	anonymous_name	is an optional identifier that is not checked by the Multics system, but is treated as if it were a person identifier.
2.	Project_id	is the identification of the user's project, which is registered by the Multics system administrator.
3.	control_args	can be chosen from the following list of control arguments:
	-brief, -bf	suppresses messages associated with a successful login.
	-no_print_off, -npf	overtypes a string of characters to provide a black area for the user to type his password; necessary only for terminals not equipped to suppress printing.

Name: explain, exp

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

Usage

explain {-long} topic1{ topic2 ... topicn}

where:

1. -long — is a control argument that specifies a long form of explain messages for given topics; if not supplied, a brief message is printed.
2. topic — is a keyword indicating the explain message desired.

Example

```
!    explain new

     02/11/76   new

     Function:  starts input of a new current file.

     Syntax:  new file_name

     Argument:  file_name is the name to be assigned to the current file.

     ready  0930

!    explain teach
     explain:  no explain segment for "teach"
     ready  0930
```

Name: goodbye, goo

Terminates a user session and disconnects the terminal. This command is identical to the bye command.

On terminals equipped with acoustic couplers, it is necessary to hang up the telephone handset.

Usage

goodbye

Example

```
! goodbye
  Smith Multics logged out 11/07/75 1240.4 mst Fri
  CPU usage 5 sec, memory usage 16.5 units.
  hangup
```

Name: hello, hel

The hello command terminates work by one user but does not disconnect the terminal. The next user can log in immediately.

Usage

hello

Example

```
!    hello

     Smith Multics logged out 11/12/75  0830.3 mst Wed
     CPU usage 8 sec, memory usage 80.9 units.


     Multics MRX.X: Multics Service, PCO,Phoenix,AZ.
     Load = 19.0 out of 41.0 units: users = 19


!    login JBrown
     Password:
!

     You are protected from preemption until 0932.
     JBrown Design logged in 11/12/75  0832.3 mst Wed from ASCII terminal "none".
     Last login 11/11/75  0729.2 mst Tue from ASCII terminal "none".
     ready  0832
```

Name: help

The help command prints information about logging in when issued prior to a successful login. If help is issued at any other time, DFAST prints a message referring the user to the explain command.

Usage

```
help
```

Example

```
!    login JBRown
     Password:
!

     Login incorrect.
     Please try again or type "help" for instructions.

!    help
     Examples of correct login:
          login Person_id
          enterp {anonymous_name} Project_id
          enter {anonymous_name} Project_id
     Uppercase and lowercase letters are different.
     Check any typing conventions for your terminal.
     Contact (appropriate accounting office) (phone) for more help.
     Please try again.

!    login JBrown

     Password:
!

     You are protected from preemption until 0830.
     JBrown Design logged in 01/28/76  0830.3 mst Wed from ASCII terminal "none".
     Last login 01/27/76  0729.2 mst Tue from ASCII terminal "none".
     ready  0830
```

Name: ignore, ign

The ignore command discards line-numbered information in the alter file rather than merging with information already stored as part of the current file. Generally, the alter file contains all line-numbered information entered since the user last executed a command that caused the alter file to be merged with the current file, such as new, old, or replace. The contents of the alter file can be examined using the list command.

Usage

```
ignore
```

Example

```
!    new new_file
     ready  1120

!    100 new text
!    110 is in the alter
!    120 file
!    save
     ready  1120

!    200 old text is
!    210 in the current
!    120 file now
!    replace
     ready  1120

!    220 file now
!    230 and also
!    list alter

     alter   11/07/75  1121.3  mst  Fri

     220 file now
     230 and also
     ready  1121

!    ignore
     ready  1121

!    220 file today
!    replace
     ready  1121
```

ignore

```
!     lisn current
      100 new text
      110 is in the alter
      120 file now
      200 old text is
      210 in the current
      220 file today
      ready  1121
```

length

Name: length, len

The length command prints the number of words in the current file. One word is equal to four characters (including punctuation, spacing, and newline characters). If the total number of characters is not a multiple of four, the last word will contain fewer than four characters. The smallest unit of storage on Multics is a record. A record consists of 1024 words. In the example shown, the user has used one record even though only 12 words were required by the file.

Usage

```
length
```

Example

```
!     100 How many
!     110 words are
!     120 in this file?
!     length
      "no name" length = 12 words (1 record)
      ready  0707
```

Name: list, lis
listnh, lisn

The list command displays information contained in the current file alone, the alter file alone, or of the current file after merging with the alter file. In the latter case, the list command causes the merge to take place thereby clearing the alter file. The output from list is preceded by a header giving the file name and the time and date. To suppress this header, the user may use listnh with the same type of arguments.

Usage

list {file} {line_number}

where:

1. file identifies the file to be listed (current or alter).
2. line_number is any valid line number.

The effects of the various uses of list are shown below:

Form	Effect
list	prints the current file (after merging with alter file).
list line_number	prints the current file beginning at the line number given; if no such line number exists, the next higher line number is used; if the line number is greater than any line number in the file, the last line of the file is printed.
list current, list cur	prints the current file (without merging contents of alter file).
list alter, list alt	prints contents of alter file after sorting into numerical order by line number (lines containing only line numbers are retained in this case).

A line number may be specified with either current or alter (e.g., list alt 40). The printout adheres to the rule given for the list line_number form above but is restricted to the file specified.

Example

The output of the listings below assumes the following contents for the current and alter files.

current file	alter file
100 text	120 new text
110 to be	150 may also
120 listed	160 be
130 next	170 listed

```
!    list current

     current      11/07/75   1214.6 mst Fri

     100 text
     110 to be
     120 listed
     130 next
     ready  1214

!    list alters 200

     alter        11/07/75   1215.2 mst Fri

     170 listed
     ready  1215

!    list

     no name      11/07/75   1216.1 mst Fri

     100 text
     110 to be
     120 new text
     130 next
     150 may also
     160 be
     170 listed
     ready  1216

!    list alter
     list:  alter segment is empty
```

Name: list_acl, la

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

Usage

list_acl {file_name} {User_ids} {-control_args}

where:

1.	file_name	identifies the file whose ACL is to be listed. If it is omitted, the home directory is assumed and 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 file/directory."
	-directory, -dr	lists the ACLs of directories only. The default is files and directories.

Note

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

Example

```
! 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 file in the home directory that has a two-component name with a second component of basic.

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

lists access modes for all entries on the home directory's ACL whose Project_id is Faculty and for the *.*.* entry.

login

Name: login, l

The login command is used to gain access to the Multics system. First, the user must dial the appropriate number to activate the terminal and wait until a message is printed by the answering service. The login command is actually a request to the answering service to start the user identification and process creation procedures. Therefore, this command can only be issued from a terminal connected to the answering service; that is, one that has just dialed up, or one that has been returned to the answering service after a session terminated with a hello command.

The login command requests a password from the user (and attempts to ensure either that the password does not appear at all on the user's terminal or that it is thoroughly hidden in a string of cover-up characters). The password is a string of one to eight letters and/or integers associated with the Person_id.

After the user responds with his password, the Multics system looks up the Person_id and the password in its tables and verifies that the Person_id is valid and that the password given matches the registered password. If these tests succeed, and if the user is not already logged in, the load control mechanism is consulted to determine if allowing the user to log in would overload the system.

Usage

login Person_id {-control_args}

where:

1.	Person_id	is the user's registered personal identifier. This argument must be supplied.
2.	control_args	can be selected from the following:
	-brief, -bf	suppresses messages associated with a successful login.
	-change_password, -cpw	changes the user's password to a newly given one. Multics asks for the old one before requesting the new. If the old password is correct, the new password replaces it for subsequent logins and the message "password changed" is printed. The user should not type the new password as part of the control argument.
	-no_print_off, -npf	overtypes a string of characters to provide a black area for the user to type his password (necessary only for users whose terminals do not have print-suppression capabilities).

-terminal_type XX -ttp XX	sets the user's terminal type to XX, where XX is one of the types listed for the corresponding control argument of the set_tty command.
-modes XX	sets the modes for terminal I/O according to XX. For a description of this argument, see the corresponding argument of set_tty.

Example

In the examples below, the user's password is shown even though in most cases Multics either prints a string of cover-up characters to "hide" the password or temporarily turns off the printing mechanism of the user's terminal.

Probably the most common form of the login request is to specify just the Person_id and the password as:

```
!    login Jones
     Password:
!    mypass
```

To set the tabs and crecho I/O modes so the terminal uses tabs rather than spaces where appropriate on output and echoes a carriage return when a line feed is typed, type:

```
!    login Jones -modes tabs,crecho
     Password:
!    mypass
```

To change the password from mypass to newpass, type:

```
!    login Jones -cpw
     Password:
!    mypass
     New Password:
!    newpass
     Password changed.
```

Name: nbrief, nbr

The nbrief command restores DFAST-issued ready messages and list command headers suppressed by a prior execution of the brief command.

Usage

```
nbrief
```

Example

```
!    brief
!    list alter
     100 random text
     110 to list
!    nbrief
     ready  1401

!    list alter

     alter     12/2/75     1210.2   mst  Mon

     100 random text
     110 to list
     ready  1401
```

Name: new

The new command starts input of a new current file. The previous current file and the contents of the alter file when the new command is issued are deleted.

Usage

```
new {file_name}
```

where file_name is the name to be assigned to the current file. (See "File Naming Conventions" in Section III for a description of valid file names.)

Example

```
!    new
     enter name:  !   newfile.basic
     ready  1301

!    100 The current
!    110 file is
!    save
     ready  1301

!    new another
     ready  1302

!    100 This is
!    110 different
!    list current

     current      11/07/75    1302.3 mst Fri

     100 This is
     110 different
     ready  1302
```

old

Name: old

The old command retrieves a file that has previously been saved either in the user's home directory or another directory to which the user has access. If the retrieval is successful, the saved file replaces the current file and the alter file is cleared. If the saved file's name includes a language component, the system is changed to that language. Otherwise, the message "enter system:" is printed and the user can type basic, dbasic, or fortran.

Usage

```
old {file_name} {system_name}
```

where:

1. file_name — is the name of a saved file; if it is not supplied, DFAST requests that the user type it in.
2. system_name — sets the current system to basic, dbasic, or fortran.

Example

```
!    system basic
     ready  0102

!    old
     enter name: ! test.basic
     ready  0102

!    old tst.fortran
     system changed to fortran
     ready  0103

!    tty
     name = tst.fortran,  system = fortran,  user = Smith.Des,  line = tty112
     ready  0103

!    old >udd>Faculty>Jones>test.basic
     system changed to basic
     ready  0103

!    tty
     name = tst.basic,  system = basic,  user = Smith.Des,  line = tty112
     ready  0103
```

Name: onecase, one

Sets the printing mode to uppercase only. At login, the mode is twocase. See "Case Conventions" in Section I. To reset the printing mode, use the twocase command.

Usage

```
onecase
```

Example

```
!    onecase
!    new newfile
     READY  1201

!    100 lowercase
!    110 text
!    lisn
     100 LOWERCASE
     110 TEXT
     READY  1201
```

rename

Name: rename, ren

The rename command assigns a new name to the current file.

Usage

rename file_name

where file_name is the name to be assigned. The name must adhere to the rules given in "File Naming Conventions" in Section III.

Example

```
!    rename test>basic
     rename:  illegal character in name
     ready  1202

!    rename
     rename:  name missing
     ready  1202

!    rename test.basic
     ready 1202
```

Name: replace, rep

The replace command saves the contents of the current file in place of the contents of a previously saved file. If the file_name argument is supplied, the current file is saved under that name regardless of the current name. If no argument is supplied, the current name is assumed and the current file replaces information previously saved under that name. If no saved file exists under either name, an error message is issued.

Usage

```
replace {file_name}
```

where file_name is the name of a saved file. If file_name is not supplied, the current name is assumed.

Example

```
!    replace
     ready  1404

!    replace test.basic
     ready  1404
```

Name: run

The run command causes the current file to be executed. The file must begin with a main program. It may be in source or object form. If the current file is an object program, it will be directly executed. If the system_name argument is supplied, the current system is changed accordingly. The contents of the current file are unaffected.

If the current file (or any external subprogram file that it calls) is in source form, it is compiled to produce a temporary object program, which is then executed. An external file must have been specified in a BASIC or FORTRAN library statement within the user's program.

Usage

```
run {system_name}
```

where system_name can be basic, dbasic, or fortran.

Example

```
!    old test.basic
     ready  907

!    run
     Your program types this
     when it runs.
     ready  907
```

Name: save, sav

The save command saves the current file either in the user's home directory or in a specified directory. If no argument is supplied, the file is saved under the current name in the home directory. If a pathname is given, the file is saved under the name given and in the directory given; the current name is unaffected.

Usage

save {file_name}

where file_name identifies the file that is to be saved; if it is to be in any directory other than the home directory, a pathname must be supplied.

Example

```
! tty
  name = "no name",  system = basic,  user = Roy.Des,  line = tty112
  ready  0620

! save >udd>ProjA>Roy>prog.fortran
  ready  0620

! tty
  name = "no name",  system = fortran,  user = Roy.Des,  line = tty112
  ready  0620

! old prog.fortran
  ready  0620

! tty
  name = prog.fortran,  system = fortran,  user = Roy.Des,  line = tty112
  ready  0621

! rename oldprog.fortran
  ready  0621

! save
  ready  0621
```

Name: scratch, scr

The scratch command empties either the current and alter files or a saved file. The current name and system are not affected. If a saved file is scratched, its name is retained in the specified directory but its contents are deleted. In this case the current and alter files are not affected. To delete the name plus the contents, the unsave command is used.

Usage

```
scratch {file_name}
```

where file_name is the name of a file saved in the home directory or some other directory to which the user has deletion privileges.

Example

```
!    tty
     name = test.basic,  system = basic,  user = Smith,  line = tty112
     ready  0730

!    scratch
     ready  0730

!    list current
     list:  current file is empty
     ready  0730

!    list alter
     list:  alter file is empty
     ready  0730

!    tty
     name = test.basic,  system = basic,  user = Smith,  line = tty112
     ready  0731
```

Name: set_acl, sa

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

Usage

set_acl file_name mode1 {User_id1 ... moden User_idn}

where:

1. file_name — is the file whose ACL is to be affected. The star convention can be used.
2. modei — is a valid access mode. This can be any or all of the letters rew. Use null, "n" or "" to specify null access.
3. User_idi — 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 modei. (For a description 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 modei has no User_id following it, the user's Person_id and current Project_id are assumed.

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 matching of access control name arguments 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 ".*.*".

Example

```
!    set_acl *.basic rew *
```

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

```
!    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 specifies properties of the user's terminal. It is needed only in those rare cases when Multics does not recognize the terminal being used at login.

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:

TTY37, tty37	device similar to Teletype 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 1200

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. A subset of modes the DFAST user may need to set are given below. Other modes are, however, supported. A full set of modes is printed with the -print control argument. Valid mode names are:

lln	where n is an integer ($10 \leq n \geq 255$) specifying the length (in character positions) of a terminal line.
crecho, ^crecho	crecho specifies that a carriage return is to be echoed when the user types linefeed (^crecho turns this mode off).
lfecho, ^lfecho	lfecho specifies that a linefeed is to be echoed when a carriage return is typed (^lfecho turns this mode off).
tabecho ^tabecho	specifies that the appropriate number of blanks are to be echoed when a tab is typed.

Modes not specified in XX are left unchanged. See "Notes" below.

-reset
turns off all modes that are not specifically set by the default modes string for the current terminal type.

set_tty

-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 a complete set of 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

The set_tty command performs 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.

Example

In the following example, a user of a TermiNet 300 with tabs establishes his terminal type.

```
!    set_tty -terminal_type tn300 -tabs -reset
```

In the next example, the user wants to use the linefeed key on his terminal for the newline character instead of the carriage return key. After the change, the user will type linefeed and the terminal will echo with carriage return so the carriage will be positioned for the next line.

```
!    set_tty -modes crecho
```

In the next example the user changes the line length to 60 characters. Lines that are longer than 60 characters will be continued on the following line. Lines that are continued will begin with "\c".

```
!    set_tty -modes l160
```

Name: sort, sor

The sort command arranges the current file in ascending sequence by line number. When more than one line has the same line number, the last one is retained. Lines that are not numbered are deleted. Text in the alter file is merged before the sort is executed. Since normal line-numbered input is automatically sorted, the sort command is applicable only to files that have been created in some other way (such as by a user program execution or with the build command).

Usage

```
sort
```

Example

```
!    old results
     ready  0915

!    lisn
     100 new data
     entered for
     100 a user
     program
     120 a user's
     130 program
     10  This is
     ready  0916

!    sort
     ready  0916

!    lisn
     10  This is
     100 a user
     120 a user's
     130 program
     ready  0916
```

Name: system, sys

The system command is used to explicitly change the current system. As described under "Command Environment" in Section III, the current system at login is basic but can be changed as a byproduct of executing various commands.

Usage

```
system system_name
```

where system_name can be basic, dbasic, or fortran.

Example

```
!    tty
     name = test,  system = fortran,  user = Smith.Design,  line = tty112
     ready  1210

!    system basic
     ready  1210

!    tty
     name = test,  system = fortran,  user = Smith.Design,  line = tty112
     ready  1211

!    compile
     ready  1211
```

Name: tty

The tty command lists the current name, current system, user identification, and terminal line numbers in the format shown below:

```
name = cur_name, system = sys_name, user = Person_id.Project_id, line = ttyn
```

Usage

```
tty
```

Example

```
!   rename data
    ready  1001

!   system fortran
    ready  1001

!   tty
    name = data,  system = fortran,  user = Smith.Design,  line = tty112
    ready  1001

!   rename datum.basic
    system changed to basic
    ready  1002

!   save
    ready  1002

!   compile
    ready  1002

!   tty
    name = datum,  system = basic,  user = Smith.Design,  line = tty112
```

twocase

Name: twocase, two

Resets the printing mode from all uppercase to mixed case. At login, this is the printing mode; thus, this command is required only after a onecase command has been previously executed. See "Case Conventions" in Section I for a description of the effects of these commands.

Usage

```
twocase
```

Example

```
!     onecase
      READY  1403

!     twocase
      ready  1403
```

Name: unsave, uns

The unsave command removes a saved file from the user's home directory or from another directory, if specified in the file_name argument. An unsave can only be successful if the user has appropriate access to the directory specified. The save command is unlike scratch, which removes the contents but leaves the file name in a directory.

Usage

```
unsave file_name
```

where file_name is the name of a saved file.

Example

```
! unsave test.basic
  ready  1620

! old test.basic
  old:  segment is not saved
  ready  1620
```

users

Name: users, use

The users command requests the number of users currently logged in under Multics. The message, as shown in the example, gives the current users and the maximum possible ("18.0/110.0") for online users and absentee users ("0/30" below).

Usage

```
users
```

Example

```
! users

Multics MRX.X, load 18.0/110.0; 18 users
Absentee users 0/30

ready  0720
```

SECTION V

TEXT EDITING

The edit command, summarized in Section IV, is used to invoke a variety of line and file editing functions. A particular function is invoked in the form of a keyword request and arguments as required, as in:

```
edit delete 100,130,140
```

Here, the delete request takes line numbers as arguments and the specified lines are removed from the current file.

When line-number arguments are required, they must be specified in ascending numerical sequence. By convention, an unbroken series of line-number arguments can be expressed using the range notation:

line1-linen

where:

1. line1 is the beginning of the range.
2. linen is the end of the range.

Both line1 and linen, if present, are affected by the request. If line1 does not exist, the next higher number is taken to begin the range. Similarly, if linen is not present, the range ends with the last line number that does not exceed linen. For example, assume the current file contains the line numbers 10, 20, 30, 40, 50, and the range 15-45 is specified. Lines affected by the request in this case are 20, 30, and 40. The maximum number of ranges that can be specified in a single request is 16. (The maximum number of files that can be specified in an edit request using file arguments is also 16.)

For BASIC programs, edit requests that change line numbers also change internal references to affected lines. This feature does not apply to FORTRAN programs.

Detailed descriptions of all edit requests are given, in alphabetical order, in the following pages.

Request: append, app

The append request combines two or more files specified by the user. Files are concatenated in the order specified without any regard for their current line numbers. The resultant file becomes the current file and is resequenced with line numbers beginning at 100 and incremented by 10 to derive subsequent numbers. For BASIC programs (if the system name is basic or dbasic), internal references to changed line numbers are also changed. This means that lines in one file should not refer to line numbers in another file.

Usage

edit append file1 file2{ file3 ... filen}

where each filei is a file name; at least two files must be specified.

Example

```
!    new newfile.basic
     ready  1101

!    10  read x
!    20  if x=0  goto 10
!    30  print x
!    save
     ready  1101

!    new subr.basic
     ready  1101

!    10  read y
!    20  if y=0 goto 10
!    30  print y
!    40  end
!    save
     ready  1102

!    edit append newfile.basic subr.basic
     ready  1102

!    lisn
     100 read x
     110 if x=0 goto 100
     120 print x
     130 read y
     140 if y=0 goto 130
     150 print y
     160 end
     ready  1102
```

Request: delete, del

The delete request removes specified lines from the user's current file.

Usage

edit delete line1{ line2 ... linen}

where each linei is a line number or a range of lines. Numbers must be specified in increasing order.

Example

```
!    new newfile
     ready  1302

!    10  do 100 item = 1,10
!    11  call r_$u(a_num)
!    12  namt = 1000*a_num+50
!    13  i = i+1
!    14  call r_$u(w_ch)
!    15  i = w_ch*9

!    edit delete 11-13
     ready  1302

!    lisn

     10  do 100 item = 1,10
     14  call r_$u(w_ch)
     15  i = w_ch*9
```

Request: desequence, des

The desequence request removes all line numbers and a single blank immediately following each, if present, from the current file.

Usage

```
edit desequence
```

Example

```
!    new newfile
     ready  1423

!    10 ten
!    20 twenty
!    30  thirty
!    edit desequence
     ready  1424

!    lisn

     ten
     twenty
      thirty

     ready  1424
```

Request: explain, exp

The explain request prints an online description of a specified edit request. If no argument is supplied, general information about the edit command is listed. See also the explain command in Section IV.

Usage

edit explain {-long} request1{ request2 ... request3}

where:

1. -long is a control argument that specifies a long form of explain messages for given requests; if not supplied, a brief message is printed.

2. requesti can be selected from the current set of edit requests.

Example

```
!    edit explain desequence

     02/14/76  edit desequence

     Function:  removes all line numbers from current file

     Syntax:  edit desequence


     ready  0900
```

Request: extract, ext

The extract request deletes from the current file all but the line numbers specified as arguments.

Usage

edit extract line1 { line2 ...linen}

where each linei is either a single line number or a range of lines.

Example

```
!     new newfile
      ready  1111

!     10  do 100 item = 1,10
!     11  call r_$u(a_num)
!     12  namt = 1000*a_num+50
!     13  i = i+1
!     .
!     .
!     .
!     17  call r_$u(w_ch)
!     18  i = w_ch*9

!     edit extract 10,14-15
      ready  1111

!     lisn

      10  do 100 item = 1,10
      17  call r_$u(w_ch)
      18  i = w_ch*9
      ready  1112
```

Request: insert, ins

The insert request inserts files at given points in a specified file. The final result becomes the current file and is resequenced beginning with line number 100 and incremented by 10 to derive subsequent numbers. For BASIC programs (if the system name is basic or dbasic), internal references to changed line numbers are also changed.

Usage

edit insert file1 file2 line1{ file3 line2 ... filen linen}

where:

1. file1 is the file into which information is inserted.
2. file2...filen are files to be inserted.
3. line1...linen are line numbers in file1 after which the associated files are to be inserted.

Example

```
!    new file1
     ready  1300

!    10  This is
!    20  new text
!    30  and this
!    save
     ready  1300

!    new file2
     ready  1300

!    10  to be inserted
!    20  in file1
!    save
     ready  1301

!    new file3
     ready  1301

!    10  is also
!    20  inserted
!    save
     ready  1301

     edit insert file1 file2 20 file3 30
     ready  1301
```

```
!     lisn
      100 This is
      110 new text
      120 to be inserted
      130 in file1
      140 and this
      150 is also
      160 inserted
      ready  1302
```

Request: join, joi

The join request concatenates specified files in the order given. No sorting or renumbering is performed. The resulting file becomes the current file.

Usage

edit join file1 file2{ file3 ... filen}

where each filei is the name of a file to be concatenated; at least two files must be specified.

Example

```
!    new newfile
     ready  1014

!    10  goto 20
!    20  goto 30
!    save
     ready  1015

!    new file2
     ready  1015

!    10  goto 20
!    20  goto 30
!    save
     ready  1015

!    edit join newfile file2
     ready 1016

!    lisn
     10  goto 20
     20  goto 30
     10  goto 20
     20  goto 30
     ready  1016
```

Request: list, lis

The list request prints one or more lines of the current file. If no line numbers are specified, the entire file is printed. If a nonexistent line is specified for listing, an error message is printed.

Usage

edit list {line1 line2 ... linen}

where each linei is a single line or range of lines.

Example

```
!    new newfile
     ready  1520

!    10 abc
!    20 def
!    30 ghi
!    40 k
!    edit list 10
     10 abc
     ready  1520
```

Request: locate, loc

The locate request causes the current file to be searched for all occurrences of a specified text string. Each line containing a match for the string is printed. If line number arguments are supplied, the search is restricted to the lines given; otherwise the entire file is searched.

Usage

edit locate /text_string/{line1 line2 ... linen}

where:

1. / is the string delimiter. Any character except blank or tab can be used as the string delimiter so long as it does not appear in the string itself.

2. text_string is the string of characters to be matched; any character (including blank) except the delimiter may be used.

3. linei is a single line or range of lines.

Example

```
!    new sample
     ready  0707


!    210 if m>n then 260
!    220 next i
!    230 if n<>m then 260
!    240 print "ok"
!    250 stop
!    260 go to 100
!    edit locate />/
     210 if m>n then 260
     230 if n<>m then 260
     ready  0707
```

Request: merge, mer

The merge request combines two or more files according to line number sequence. The first file specified serves as the primary file for merging; that is, the file into which all other specified files will be merged. Lines from subsequent files are inserted into the primary file in the proper numerical sequence. If duplicate lines occur, the last one encountered during the merge is retained. The resulting file becomes the current file.

Usage

edit merge file1 file2{ file3 ... filen}

where each filei specified is merged into file1.

Example

```
!    new filea
     ready  1430

!    10  Primary file
!    40  to be merged
!    60  with others
!    save
     ready  1430
!    new fileb
     ready  1430

!    20  secondary file
!    30  to be merged
!    40  with filea
!    save
     ready  1431

!    edit merge filea fileb
     ready  1431

!    lisn
     10  Primary file
     20  secondary file
     30  to be merged
     40  with filea
     60  with others
     ready  1431
```

Request: move, mov

The move request relocates specified lines within the current file to a given location. Relocated lines are placed after a specified line number and assigned new line numbers by incrementing that value by one. For example, if three lines are moved to line 100, they will be given the line numbers 101, 102, and 103. If a sequence of lines is moved so that their numbers would not fit between the line specified and the line originally specified, succeeding lines are resequenced with an increment of one until there is no overlap.

Usage

```
edit move line1 line2
```

where:

1. line1 is a line or range of lines to be moved.
2. line2 is the line after which line1 will be inserted.

Example

```
!    new newfile
     ready  1300

!    10  ten
!    20  twenty
!    30  thirty
!    40  forty

!    edit move 40 20
     ready  1300

!    lisn
     10  ten
     20  twenty
     21  forty
     30  thirty
     ready  1301

!    3   three
!    7   seven
!    9   nine
!    10  ten
!    11  eleven
!    edit move 8-11 21
     ready  1301
```

```
!     lisn
      3    three
      7    seven
      20   twenty
      21   forty
      22   nine
      23   ten
      24   eleven
      30   thirty
      ready  1301
```

Request: prefix, pre

The prefix request inserts a given character string immediately before each occurrence of an existing character string. Line numbers are not affected.

Usage

```
edit prefix /old_string/new_string/line1{ line2 ... linen}
```

where:

1.	/	is any delimiter except blank or tab; the delimiter character cannot be a character in either old_string or new_string.
2.	old_string	is the string to be located.
3.	new_string	is the string to be inserted.
4.	linei	is a single line number or range of lines; each linei specifies the bounds within which the substitution is to occur.

Example

```
!    new new_file
     ready  1407

!    10 let a = 10
!    20 let b = 100
!    30 let c = 1000
!    edit prefix /100/0/0-40
     ready  1407


!    lisn

     10 let a = 10
     20 let b = 0100
     30 let c = 01000
     ready  1407
```

Request: replace, rep

The replace request substitutes a given character string within a specified line or range of lines. Line numbers are unaffected.

Usage

edit replace /old_string/new_string/line1{ line2 ... linen}

where:

1.	/	is any delimiter except blank or tab; the delimiter character cannot be a character in either old_string or new_string.
2.	old_string	is a string of characters to be located.
3.	new_string	is a string of characters to be substituted for each occurrence of old_string within the range given.
4.	linei	is a single line number or range of lines; each linei specifies the bounds within which the substitution is to occur.

Example

```
!    new new_file
     ready  1101

!    100 1 January 1975
!    110 1 February 1975
!    120 1 March 1975
!    edit replace /5/6/100-120
     ready  1101


!    lisn
     100 1 January 1976
     110 1 February 1976
     120 1 March 1976
     ready  1101
```

Request: resequence, res

The resequence request renumbers specified lines in the current file, beginning with a given line number and adding a given increment to derive subsequent numbers. If only a beginning line is given, resequencing continues to the end of the file. If a range of lines is given, resequencing terminates at the upper bound of the range. If no argument is given, the default assumption is to begin renumbering at the beginning of the file (denoted by 0), to assign 100 as the first line number, and to derive subsequent numbers in increments of 10. For BASIC programs (if the system name is basic or dbasic), internal references to changed line numbers are also changed.

Usage

```
edit resequence {new_num, start_line, inc}

edit resequence new_num, range, inc
```

where:

1. new_num — is the first new line number to be assigned (100 by default).
2. start_line — is the line to which new_num is to be assigned (0 by default).
3. inc — is the increment used to derive subsequent line numbers (10 by default).
4. range — is a range of lines delimiting the resequencing operation.

Example

```
!    new newfile
     ready  1301

!    210 if m>n then 260
!    220 next i
!    230 if n<>m then 260
!    240 print "ok"
!    250 stop
!    260 go to 400
!    edit resequence
     ready  1301

!    lisn
     100 if m>n then 150
     110 next i
     120 if n<>m then 150
     130 print "ok"
     140 stop
     150 go to 400
     ready  1301
```

```
!    edit resequence 210 110-130 5
     ready  1302


!    lisn
     100 if m>n then 150
     210 next i
     215 if n<>m then 150
     220 print "ok"
     140 stop
     150 go to 400
     ready  1302
```

Request: sequence, seq

The sequence request adds a new set of line numbers to the current file, beginning with a given line number and adding a given increment to derive subsequent numbers. If the file already has line numbers, these are retained but become part of the text on the line. If no increment is supplied, 10 is assumed. If no arguments are supplied, the first line number in the file will be 100.

Usage

```
edit sequence {first_num inc}
```

where:

1.	first_num	is the first line number (100 by default).
2.	inc	is the increment used to derive subsequent numbers (10 by default).

Example

```
!    build
!    nonnumbered
!    file
!    input
!

     ready  1503

!    edit sequence
     ready  1503

!    lisn
     100 nonnumbered
     110 file
     120 input
     ready  1503

!    edit sequence 500 5
     ready  1504

!    lisn
     500 100 nonnumbered
     505 110 file
     510 120 input
     ready  1504
```

Request: string, str

The string request converts the current file into a random-access string file. Each input line, including its line number, is converted into a separate string and the newline character(s) are removed.

Usage

edit string n

where n is a number giving the maximum length of any string to be used.

Request: suffix, suf

The suffix request inserts a given character string immediately following each occurrence of an existing character string. Line numbers are not affected.

Usage

edit suffix /old_string/new_string/line1{ line2 ... linen}

where:

1.	/	is any delimiter except blank or tab; the delimiter character cannot be a character in either old_string or new_string.
2.	old_string	is the string to be located.
3.	new_string	is the string to be inserted.
4.	linei	is a single line number or range of lines; each linei specifies the bounds within which the substitution is to occur.

Example

```
!    lisn

     100  I am
     110  go
     120  to the
     130  store
     ready  1300

!    edit suffix /go/ing/110
     ready  1300

!    lisn 110
     110 going
     ready  1300
```

APPENDIX A

COMMAND SUMMARY

The summary below is in alphabetical order by command name. For summary descriptions organized by function, see "Command Repertoire" in Section III.

append	appends unsorted contents of alter file to current file.
bill	prints accounting information.
brief	establishes brief output mode.
build	initiates mode of input for nonnumbered lines.
bye	terminates a user session and disconnects the terminal.
catalog	prints information about files stored in specified directories.
compile	compiles source code in current file.
delete_acl	removes an entry from an access control list (ACL).
dprint	queues a file for printing on the high-speed line printer.
edit	requests specified DFAST text-editing operations.
enter, enterp	logs in anonymous user.
explain	prints online description of specified topic.
goodbye	terminates a user session and disconnects the terminal.
hello	terminates a user session but leaves the terminal connected for subsequent user.
help	prints online description of login procedures.
ignore	discards contents of the alter file.
length	prints the number of words in the current file.
list, listnh	lists all portions of the current and/or alter files (listnh suppresses header information).
list_acl	prints an entry in an access control list (ACL).
login	connects registered user to Multics; used at dialup or after a hello command.
nbrief	terminates brief output mode.
new	initiates a new current file, deletes both the current and alter files and changes the current name.
old	retrieves a previously saved file and makes it the current file.

onecase	establishes a single-case input/output mode.
rename	renames the current file.
replace	replaces the contents of a previously saved file with the contents of the current file.
run	compiles, if necessary, and executes the current file.
save	stores the current file.
scratch	empties both the current and alter files.
set_acl	adds or changes an entry in an access control list (ACL).
set_tty	modifies terminal type and modes associated with user's terminal.
sort	sorts the current file into ascending line-numbered sequence.
system	resets the current system (compiler).
tty	prints current command environment.
twocase	establishes two-case input/output mode.
unsave	deletes a stored file.
users	prints the number of users currently active on the entire Multics system.

APPENDIX B

DFAST BASIC

DFAST BASIC is the same as standard Multics BASIC (as described in Multics BASIC, Order No. AM82) with the exceptions stated below.

1. The library statement. External files containing subprograms called by the programs in the user's current file must be listed in a library statement in the calling program.

   The library statement has the form:

   library "file1"{,"file2",...,"filen}

   The library statement lists the names of files containing the subprograms to be used. The names are enclosed in quotation marks and separated by commas. If only the filename is given in a library statement, it is located in the home directory at execution time.

2. The setdigits statement. The setdigits statement dynamically controls the number of digits in a numeric value that may be printed as output. It has the form:

   setdigits formula

   The value expressed by the formula in the statement is truncated to its integer value and represents the number of print columns that will be utilized by all subsequent print statements until another setdigits statement is executed or until program execution terminates. From 1 to 19 printed columns may be specified.

   In addition to the specified number of digits, the sign of the number is printed. An exponent is also printed if all digits to the left of the decimal point cannot be contained in the number of digits expressed by the formula. The setdigits statement is valid only for double precision programs.

3. The characters "-" and "." are allowed in subprogram names.

4. A $ used in a format statement as a field delimiter need not be followed by "+" or "-"; "-" is assumed.

5. The asc function recognizes the abbreviation "apo" to mean apostrophe.

6. The rules about the Multics environment and non-BASIC programs (Section XIII and Appendix B of the Multics BASIC manual, Order No. AM82) are replaced by the rules for DFAST.

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

HONEYWELL INFORMATION SYSTEMS

Technical Publications Remarks Form

CUT ALONG LINE

TITLE: SERIES 60 (LEVEL 68) MULTICS DFAST SUBSYSTEM USERS' GUIDE

ORDER NO. AT59, REV. 0

DATED MARCH 1976

ERRORS IN PUBLICATION

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Your comments will be promptly investigated by appropriate technical personnel and action will be taken as required. If you require a written reply, check here and furnish complete mailing address below. ☐

FROM: NAME _____ **DATE** _____

TITLE _____

COMPANY _____

ADDRESS _____

_____

CUT ALONG L

FOLD ALONG LINE

PLEASE FOLD AND TAPE –
NOTE: U. S. Postal Service will not deliver stapled forms

FIRST CLASS
PERMIT NO. 39531
WALTHAM, MA
02154

Business Reply Mail
Postage Stamp Not Necessary if Mailed in the United States

Postage Will Be Paid By:

HONEYWELL INFORMATION SYSTEMS
200 SMITH STREET
WALTHAM, MA 02154

ATTENTION: PUBLICATIONS, MS 486

FOLD ALONG LINE

Honeywell

Honeywell

Honeywell Information Systems
In the U.S.A.: 200 Smith Street, MS 486, Waltham, Massachusetts 02154
In Canada: 2025 Sheppard Avenue East, Willowdale, Ontario M2J 1W5
In Mexico: Avenida Nuevo Leon 250, Mexico 11, D.F.

20992, 5C678, Printed in U.S.A.

AT59, Rev. 0