SERIES 60 (LEVEL 68)

MULTICS FAST SUBSYSTEM REFERENCE MANUAL

SUBJECT

Reference Manual for the Multics FAST Subsystem Containing Information for Users of the FAST Subsystem Within the Multics Storage System

SPECIAL INSTRUCTIONS

This manual completely supersedes Rev. 0 of the *Multics FAST Subsystem Users' Guide* dated March 1976.

Section 5 of this manual has been completely revised to reflect the current version of all Multics commands presented. No change bars appear in Section 5; however, all other sections of this manual contain change bars to indicate information changed since publication of the original document.

SOFTWARE SUPPORTED

Multics Software Release 8.0

ORDER NUMBER

AU25-01

September 1979

Honeywell

PREFACE

This document describes the Multics FAST subsystem, a time-sharing facility supporting BASIC and FORTRAN program development. The command language and repertoire of this subsystem are based on that of the full Multics system.

The manual presupposes no knowledge of the Multics system however, the new user may find the New Programmers' Introduction to Multics, Order No. AL40, particularly helpful. BASIC programmers using this subsystem are referred to the Multics BASIC manual (Order No. AM82). FORTRAN programmers are referred to the Multics FORTRAN Reference Manual (Order No. AT58), and the Multics FORTRAN Users' Guide (Order No. CC70).

This manual is intended to permit the programmer to use the Multics FAST subsystem immediately. The introductory information in Section 1 and the sample session in Section 2 provide enough information to begin using the system. An overview of the storage system is given in Section 3. Section 4 describes Access Control, and Section 5 contains the full command repertoire.

File No.: 1L13

AU25-01

CONTENTS

SECTION 1

INTRODUCTION

The Multics FAST subsystem is an easy-to-use, time-sharing facility designed primarily for creating and running BASIC and FORTRAN programs. A simplified command language is used to create and edit text files and to compile and run programs.

The segments and programs of this subsystem are part of the Multics system environment. However, all the information required to use the subsystem is contained in this manual; no prior knowledge of Multics is needed.

SUBSYSTEM FEATURES

BASIC is the same BASIC used in the full Multics system. It was based on the Darthmouth BASIC. See the Multics BASIC manual, Order No. AM82, for information on programming in BASIC on Multics.

FORTRAN is a superset of ANSI FORTRAN with extensions for compatability with Multics. A number of time-sharing oriented features have been added and the use of expressions in language constructs generally expanded. The FORTRAN language used on the Multics FAST subsystem is described in the Multics FORTRAN Reference Manual, Order No. AT58.

Command level text editing facilities on this subsystem support editing and sorting of line-numbered input. Unnumbered text can be created and modified with the edm command. File handling facilities support segment creation, deletion, modification, and renaming. A user can access any segment 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 that are from system libraries.

A variety of online information is available to a user on request. This includes brief descriptions of commands, information on the current state of Multics, and segment-related information. (See the help command in Section 5 for more information.)

SEGMENT NAMING CONVENTIONS

A segment is the basic unit of information in the Multics storage system (see Section 3, "Multics Storage System," for details). Each segment has access attributes (described in Section 4, "Access Control"), at least one name, and may contain data or programs.

A segment name is called an entryname. A user must construct an entryname for each segment he creates. An entryname must be from one to 32 characters long. It can contain any uppercase or lowercase alphabetic character, any number (0-9), and the characters hyphen (-), underscore (_), and period (.). A period has a special effect, dividing an entryname into separate components that are interpreted by the Multics system. For example, the use of the period in:

```
test.fortran
```

produces a two-component entryname whose second component is a language suffix indicating that the segment is a FORTRAN source program. All user-written FORTRAN and BASIC source programs should follow this convention and end with the appropriate suffix: basic for BASIC programs and fortran for FORTRAN programs. When a source program is compiled, the segment name of the object code is the same as the segment name of the source program minus the suffix. For example, when the source programs named:

```
alpha_text.basic  and  alpha_text.old.basic
```

are compiled, object code segments named:

```
alpha_text  and  alpha_text.old
```

are created.

Multiple Names

A user can give a segment more than one entryname and can refer to the segment by any one of its names. The original name of a segment is considered the primary name (the main name associated with the segment).

Star Names

Many commands that accept an entryname argument allow the entryname argument to be a star name. A star name is an entryname that contains an asterisk (*). According to the star convention (described in detail in Section 3 of the Multics Programmers' Manual (MPM) Reference Guide, Order No. AG91), a star name is matched with entrynames in a single directory to identify a group of entries. Each asterisk in a star name component matches any corresponding component of an entryname. For example:

*	identifies all one-component entries.
.	identifies all two-component entries.
*.basic	identifies all two-component entries that have basic as their second component.
beta.*.*	identifies all three-component entries whose first component is beta.

A double asterisk (**) can be used to match any number of corresponding components (including none) of an entryname. The double asterisk alone can be used to indicate all entrynames in the directory. For example:

**	identifies all entrynames regardless of the number of components.
blue.**	identifies all entrynames (regardless of the number of components) whose first component is blue.

one.**.two identifies all entrynames (regardless of the number of components) whose first component is one and whose last component is two.

Equal Names

Many commands that require entryname arguments to be given in pairs allow the first argument of the pair to be a star name and the second argument, an equal name. If the first entryname argument is a star name, the second entryname argument must be an equal name. According to the equal convention (described in detail in Section 3 of the MPM Reference Guide), an equal name contains characters that represent one or more components of the entrynames identified by a star name. Each equal sign (=) in an equal name represents the corresponding component of an entryname identified by a star name. For example:

```
rename  *.data_base  =.data
```

renames all two-component entrynames with data_base as their second component so these entrynames have, instead, a second component of data.

Each double equal sign (==) component of an equal name represents one or more components of an entryname identified by a star name. For example:

```
add_name  **.basic  ==.old.basic
```

adds a name to all entrynames with a last component of basic. The last two components of these new entrynames are old.basic, and the first components are the same as those of the entrynames ending in basic.

COMMAND LANGUAGE CONVENTIONS

Multics FAST subsystem commands are invoked when the user is at command level (i.e., after he logs in or when a command completes, encounters an error, or is stopped by issuing the quit signal). Command level is normally indicated by a ready message (explained under "Logging In" in Section 2).

A command invocation consists of a command procedure name (command name) plus any character-string arguments to be passed to the command procedure when the command is invoked. A command line can begin at any horizontal position. When arguments are supplied, at least one blank or tab must separate them from the command name. Arguments are separated from each other by blanks or tabs and the entire command line is terminated by a newline character.

TYPING CONVENTIONS

Typing errors can be corrected using the special characters # and @. Although both characters are printed graphics, they do not become part of the line. The number sign (#) erases itself and the contents of the previous print position. Several successive number signs erase an equal number of graphics. One erase character typed immediately after "white space" (any combination of tabs and spaces) causes the entire white space to be erased. For example, typing:

```
print_text              nswf##### newfile
```

produces:

```
print_text newfile
```

The commercial at sign (@), erases the contents of the entire line up to and including itself. For example, typing:

```
print_test n@print_text newfile
```

produces:

```
print_text newfile
```

If a programmer needs the number sign or at sign to appear in his segment, he can type either one preceded by a backslash (\) or a cent sign (¢) depending on the terminal type. For example:

```
\# and \@
```

print as:

```
# and @
```

QUIT SIGNAL

The user can interrupt the subsystem during command or program execution or while editing by depressing the ATTN, INTERRUPT, BRK, or QUIT button on the terminal. This action may be necessary because a program is in a loop or because the user issued a print request by mistake while editing using the edm editor. If the user interrupts while using the edm editor, the system queries, "Do you want to continue editing?". If the user answers "yes", he is returned to the editor. In all other cases, the FAST subsystem returns to command level and issues a ready message.

ERROR HANDLING

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

```
command_name: message
```

Several commands can invoke the same error message. For example, "unknown argument" can be issued for most commands. When a subsystem error occurs, the user is issued a new ready message and can reissue 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 help command (described in Section 5).

SECTION 2

USING THE MULTICS FAST SUBSYSTEM

LOGGING IN

Before the user can log in, he must be registered under a project associated with the Multics FAST subsystem. When he is registered, he is assigned a unique identification (called a Person_id) and a password, both of which must be entered precisely as assigned whenever he logs in. For example, if the user's Person_id is JBrown, he cannot log in as Jbrown or J Brown; if his password is showboat, he cannot use Showboat or show boat. To ensure confidentiality, and depending on the terminal type, the Multics system either turns off the printing mechanism or prints a string of cover-up characters so the user's password cannot be read by others.

After the user's Person_id and password have been successfully entered, the Multics initial message is typed and the subsystem issues a ready message of the form:

```
r 0920
```

where 0920 is the current time. This message is printed throughout the session to inform the user that the subsystem has completed a specified task and is again ready to accept user input.

A sample login, including the messages printed by the Multics system and the FAST subsystem, is shown below. Prior to this interchange, the user must dial the appropriate telephone number to establish a connection with the Multics system. The exclamation point (!) is used here and throughout this document to denote text typed by the user. The exclamation point is simply a notational convention and should not actually be typed. Comments (not part of the session) are to the right and preceded by the slash character (/).

```
     Multics MRX.X: Multics Service, PCO, Phoenix, AZ.
     Load = 26.0 out of 100 units: users = 26
!    login JBrown
     Password:
!    showboat          /in an actual session, the password will either
                       /not print or will be obscured by cover-up characters

     You are protected from preemption until 0829.
     JBrown Demo logged in 09/19/79  0729.2 mst Wed from ASCII terminal "H17".
     Last login 09/18/79  1230.0 mst Tue from ASCII terminal "H16".
     r 0730
```

Notice the three-line message printed by the Multics system after the user types his password. The second line states the Person_id (JBrown) and Project_id (Demo) of the user, the date (09/19/79) and time (0729.2 mst) of log in, the day of the week (Wed), and the terminal from which the user is logged in (ASCII terminal "H17"). The third line states the date (09/18/79), time (1230.0 mst), day (Tue), and terminal (ASCII terminal "H16") of the user's last login. Any other important information that should be given to all users can be included at this point, e.g., scheduled computer time over a holiday.

Creating and Leaving the FAST Subsystem

There are two methods the user may employ to enter the FAST subsystem. If the user goes through the usual Multics login procedure described above, after receiving the three-line message printed by the Multics system, the user is at Multics command level. The user then types the Multics command:

```
! fast
```

to enter the FAST susbystem.

Alternatively, instead of going through the entire Multics login procedure, the user, when logging in, types:

```
! login JBrown -po fst_process_overseer_
```

and he will be in the FAST subsystem.

In either event, when the user has finished working in the FAST subsystem, to leave the subsystem the user types:

```
! quit
```

This command returns the user to Multics command level. The user may then invoke other Multics commands or proceed to log out by typing:

```
! logout
```

as described in "Logging Out," below.

COMMAND LEVEL

Having successfully logged in, the user is at command level in his working directory and can either invoke a subsystem command or input temporary text. All text input at command level is called temporary text and must begin with a line number. 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. Blanks or tabs preceding line numbers are ignored. All of the following lines will be entered into temporary text:

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

Temporary text can be saved and compiled and executed, or simply saved, and then the contents of a new segment can become the temporary text.

The following commands are used in the sample session below. They are very easy to learn and the only commands needed to begin programming on the Multics FAST subsystem. The entire command repertoire is described in Section 5.

new	deletes the temporary text and creates an entryname for the new temporary text.
old	replaces the temporary text with the contents of a previously saved segment.
print_text	prints all or portions of the temporary text.
run	compiles and executes the temporary text or a specified segment.
save	saves the temporary text in the segment specified.
basic	creates a BASIC object segment.
fortran	creates a FORTRAN object segment.
list	lists segments stored in a specified directory.

LOGGING OUT

When a user has completed a session, he must log out. To log out and disconnect the terminal, he must issue the logout command, wait for the logout message, and disconnect the acoustic coupler.

SAMPLE SESSION

The following session shows the compilation and execution of a BASIC program and the compilation of a FORTRAN program. Full descriptions of the commands used in the sample session are given in Section 5.

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 1015.
     Smith Design logged in 09/10/79  0915.3 mst Mon from ASCII terminal "H17".
     Last login 09/09/79  1145.0 mst Sun from ASCII terminal "H16".
     r 0916
```

After entering the FAST subsystem, to begin entering input the user issues the subsystem command new, and supplies an entryname for the program being created. Any name that adheres to the naming conventions specified in "Segment Naming Conventions" in Section 1 can be assigned. This user wants to create a BASIC program so the language suffix is basic.

```
!    new sum.basic
     r 0917
```

Input of the source code begins now. This program adds two numbers and prints the sum.

```
!     10 input x, y
!     20 lt z = x + y
!     30 print z
!     20 let z = x + y          /user corrects line 20 by retyping it
!     print_text
      10 input x, y
      20 let z = x + y
      30 print z
      r 0918
```

The user compiles and executes the program using the run command. Notice that the run command is invoked here without an argument. Therefore, the temporary text is used.

```
!     run
      No end statement as of 30          /BASIC error message
      run: 1 error found.  no execution.  /subsystem error message
      r 0919
```

The user adds the missing end statement and types run. This time the program executes correctly. It is a feature of the BASIC language to prompt the user for input with the question mark (?) character. In the example below, when the user is prompted, he types in the numbers 12 and 78.

```
!     40 end
!     run
      ? 12
      ? 78
      90
      r 0920
```

The user decides to change the printout to include the expression "sum =". This change is made by retyping the line.

```
!     30 print "sum =", z
!     run
      ? 12
      ? 78
      sum =           90
      r 0921
```

In the next portion, the user saves the source program that he has completed.

```
!     save           /the program is saved with the name sum.basic
      r 0922
```

The user compiles the source program so that the next time he wishes to run the program, he will not pay the cost of compilation.

```
!    basic sum                  / this creates an object segment named sum
                                / in the working directory
     r 0923
```

To make sure the object segment works, he tests the program again. This time he types the name of the program with the run command. When this is done, the run command looks in the working directory for a segment named sum that contains object code.

```
!    run sum                    / this runs an object segment
                                / that is in the working directory
     ? 2
     ? 3
     sum =          5
     r 0924
```

The user lists the segments in his working directory. It currently contains four segments: sum.basic contains the source code just entered, sum contains object code, test.fortran and multiply.fortran are previously entered programs.

```
!    list

     Segments = 3, Lengths = 4.

     re      1  sum
     r w     1  sum.basic
     r w     2  test.fortran
     r w     1  multiply.fortran

     r 0925
```

The user makes the previously created segment, multiply.fortran, the temporary text.

```
!    old multiply.fortran
     r 0926
```

He prints the segment of temporary text.

```
!    print_text

     multiply.fortran 03/01/76   0926.3 mst Mon

     10 input 100, x,y
     20 z = x*y
     30 print 100,z
     32 100 format (f5.0,f5.0)
     40 end
     r 0927
```

The user compiles the source program so that the next time he wishes to run the program, he will not pay the cost of compilation.

```
!    fortran multiply
     r 0928
```

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

```
!     logout
      Smith Design logged out 09/10/79 0929.2 mst Mon
      CPU usage 5 sec, memory usage 16.5 units.
```

SECTION 3

MULTICS STORAGE SYSTEM

The basic unit of storage in the Multics storage system is a segment. Segments are cataloged in directories and organized in a tree-structured hierarchy to form the Multics storage system. Figure 3-1 shows a portion of this structure in a very simplified form.

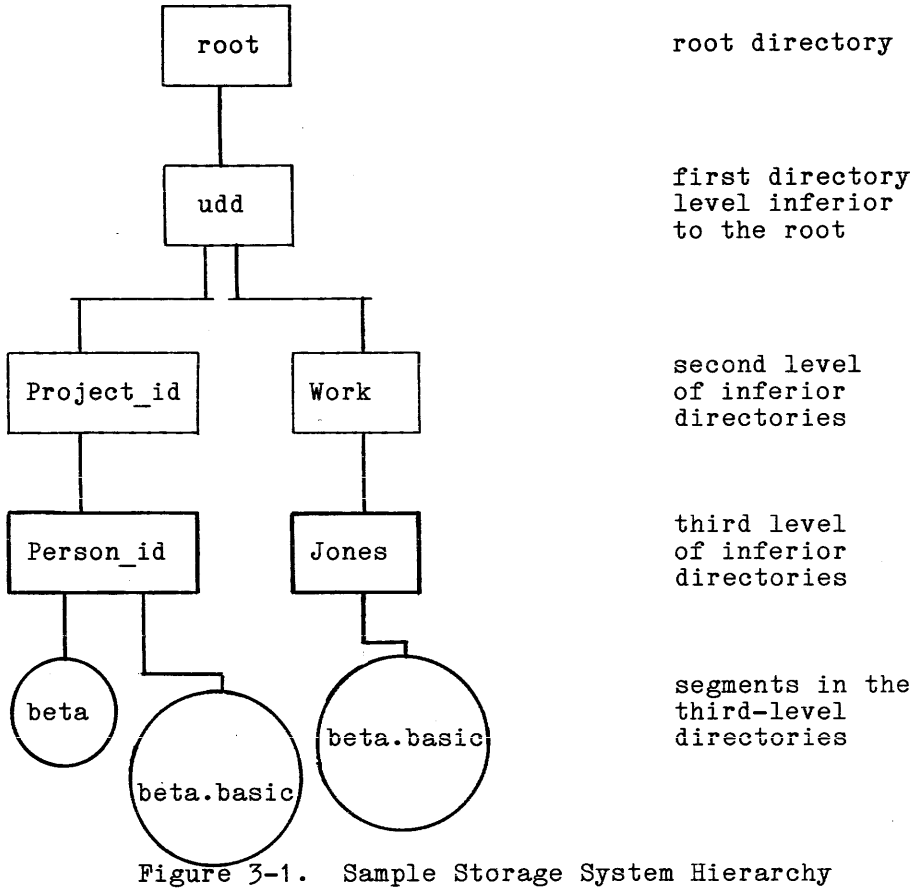

Figure 3-1. Sample Storage System Hierarchy

ENTRYNAMES

Segments, directories, and links in the storage system are known as entries. Each entry has an entryname that is constructed according to the rules stated under "Segment Naming Conventions" in Section 1.

PATHNAMES

A pathname is a sequence of entrynames that specifies the position of an entry in the directory hierarchy by tracing a series of directories from the root to the desired entry. By convention, individual entrynames in a pathname are separated by the greater-than character (>) and the root directory need not be specified. The following pathnames indicate the position of three segments in the third-level directories of Figure 3-1:

```
>udd>Project_id>Person_id>beta

>udd>Project_id>Person_id>beta.basic

>udd>Work>Jones>beta.basic
```

Notice the directories Person_id and Jones both contain segments named beta.basic. Different directories can contain segments with the same entryname; however, within a single directory, entrynames must be unique.

WORKING DIRECTORY

After successfully logging in to the Multics system, the user is in his home directory. For users of the Multics FAST subsystem, the home directory is also the working directory and is referred to as such throughout this manual. Multics FAST subsystem users cannot change to another working directory.

A user can address any segment in his working directory simply by giving its entryname as the path argument to a command. If a user wishes to address a segment in a directory other than his working directory, he must supply the full pathname of the segment as the path argument to the command. For example (using Figure 3-1), if Jones wishes to print his program beta.basic, he types:

```
print_text beta.basic
```

If he wants to print beta.basic in the Person_id directory, he must type:

```
print_text >udd>Project_id>Person_id>beta.basic
```

LINKS

A link is an entry in a directory that points to a segment in some other directory. A link enables a user to access a segment as if it were in his working directory. Given the proper access permission, a user may address the segment linked to by specifying the link name.

SECTION 4

ACCESS CONTROL

Each segment stored in the Multics storage system has a set of access rights associated with it. The user has control of the access rights to all the segments in his working directory and can specify both those users who has access to a particular segment and the type of access. For example, a user may specify that anyone has access to read his segment but that only he himself has access to write in it. By default, other users can neither read nor write in (change) the segments in a user's working directory. So, when a user creates a new segment, no other user can have access to it unless he gives it to them.

ACCESS CONTROL LIST

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

The User_id in the ACL consists of a three-component name: Person_id, Project_id, and an instance tag, separated by periods. The Person_id is a unique name assigned to each user -- usually some form of the user's name. The Project_id is the name assigned to the project under which the user is registered on Multics. And, the instance tag is assigned by the Multics system when the user logs in. Whenever anyone tries to access a segment on the Multics system, his three-component name must match one of the entries on the ACL of that particular segment; if not, he has no access to that segment.

ACCESS MODES

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

Access modes for segments are:

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

Access modes for directories are:

status	(s)	the contents of the directory can be listed; the attributes of segments and directories contained in the directory can be obtained.
modify	(m)	the segments in the directory can be deleted; the attributes of existing segments and directories contained in the directory can be changed or deleted.
append	(a)	new segments and directories can be created in the directory.
null	(n)	access to the directory is denied.

As stated earlier, a user of the Multics FAST subsystem can assign other users access only to segments in his working directory. Once specified, the access to a segment is not permanent; the user can change it at will by specifying different modes or User_ids.

SETTING ACCESS

The command the user invokes to set the ACL, set_acl, either adds an entry to the ACL or modifies an existing entry. The set_acl command, which can be abbreviated sa, is described in detail in Section 5 and has the general format:

```
sa pathname mode(s) User_id
```

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

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

where JDoe is Jane Doe's Person_id.

If he instead decides that his segment 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 in the above command lines tells the Multics system 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 that Tom gave in both commands, JDoe.*.*, matches:

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

When the user wants to denote <u>any</u> 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_id of *.*.* specifies <u>any</u> Multics user.

LISTING ACCESS

To check the ACL of a segment, the user invokes the command that lists the ACL, list_acl. The list_acl command, which can be abbreviated la, is described in detail in Section 5 and has the general format:

```
la pathname
```

As explained earlier, any pathname that is simply an entryname indicates a segment in the working 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 system processes that control such things as printing and making copies of segments or backup tapes. The system normally places appropriate ACL entries on every segment the user creates so that system processes will have the necessary access to perform the various backup, metering, and input/output functions.

DELETING ACCESS

A third access control command, delete_acl, allows the user to delete ACL entries. This command, which may be abbreviated da, is described in detail in Section 5 and has the same general format and rules as the list_acl command.

For example, if Tom Smith has changed segment 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      Gary.Merlin.*
     re      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.*
```

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

AUTHORIZATIONS

Several authorization parameters are kept for users and projects in system and project tables.

For each person registered on the system, a person maximum authorization can be kept (by the system security administrator) in a system table. This maximum authorization is composed of the highest sensitivity level and all the categories of information this person may ever access, on any project. Each installation has its own procedures for assigning and changing person maximum authorizations.

Each person registered on the system also has a default login authorization, kept in a system table, and changeable by the person himself (see the login command description in Section 5). If a person does not specify an authorization at log-in time, the default authorization is assumed.

For each project on the system, a project maximum authorization is kept (by the system security administrator) in a system table. This maximum authorization is composed of the highest sensitivity level and all the categories of information that any user logged in on this project may ever access. Again, each installation has its own procedures for assigning and changing project maximum authorizations.

For each person on a project (i.e., for each user), a user maximum authorization is kept (by the project administrator) in a project table. This maximum authorization is composed of the highest sensitivity level and all the categories of information this person may access when logged in on this project.

At the time the user logs in, a process is created and its authorization and maximum authorization are established.

The process maximum authorization is the highest authorization the process can attain. It is computed directly from the three maximum authorization parameters:

person maximum authorization
project maximum authorization
user maximum authorization on the project

The maximum authorization has the same form as an authorization or an access class. The sensitivity level of the process maximum authorization is the smallest of the sensitivity levels of the three maximums. The category set of the process maximum authorization is composed only of categories contained in the category sets of all three maximums. Thus, the process maximum authorization is the highest authorization that is less than or equal to each of the three maximum authorization parameters.

The process authorization is computed directly from the following parameters:

process maximum authorization
terminal access class
-auth argument to login or new_proc (or default)

The sensitivity level of the process authorization is computed in the same manner as that of the process maximum authorization -- it is the smallest of the sensitivity levels of the three parameters and its category set is composed only of categories contained in the category sets of the three parameters. Therefore, the process authorization is the highest authorization that is less than or equal to each of the three parameters. If the process authorization is not less than or equal to the process maximum authorization, then the process is not created, login or new_proc fails, and the system prints a message.

SECTION 5

COMMANDS

COMMAND REPERTOIRE

A complete list of commands available on the Multics FAST subsystem is given below, organized in terms of general function. A detailed description of each of these commands, in alphabetical order, is presented in this section.

Several subsystem commands are standard Multics commands. These commands have the same capabilities on the FAST subsystem as they do on the full Multics system.

Access to the System

enter
: connects an anonymous user to the system.

login
: connects a registered user to the system.

logout
: terminates a user session and disconnects the terminal.

Edit and Print

new
: deletes the temporary text and creates an entryname for the new temporary text.

old
: replaces the temporary text with the contents of a previously saved segment.

print_text
: prints all or portions of the temporary text.

save
: saves the temporary text in the segment specified.

change
: replaces a specified character string within a line.

delete_line_numbers
: removes the line number from each line of the temporary text.

delete_text
: deletes lines from the temporary text.

locate
prints lines from the temporary text containing a specified string.

input
establishes a mode of input where the system supplies the line number and the user completes the line.

merge_text
inserts the contents of a segment into the temporary text.

move_text
relocates one or more lines of the temporary text.

resequence
changes the line numbers in the temporary text.

add_line_numbers
adds a line number to each line of the segment.

edm
invokes an editor more powerful than the command level editor; used to edit text without line numbers.

dprint
queues a segment for printing on the high-speed line printer.

Compile and Execute

run
compiles and executes the temporary text or a specified segment.

basic
creates a BASIC object segment.

fortran
creates a FORTRAN object segment.

Information

help
prints online description of specified topic.

info
prints the segment name of the temporary text, date, time, quota, money spent, and total money allotted.

hmu
prints the number of users.

ready_off
suppresses the ready message.

ready_on
causes the ready message to be printed.

Storage System

copy
: copies a segment.

list
: prints information about segments and directories.

delete
: deletes a segment.

add_name
: adds a name to a segment.

delete_name
: deletes a name from a segment.

rename
: changes the name of a segment.

link
: creates a link.

unlink
: deletes a link.

Access Control

delete_acl
: removes an ACL entry.

list_acl
: prints an ACL entry.

set_acl
: adds or changes an ACL entry.

Terminal Control

set_tty
: allows the user to change the default modes for the terminal.

COMMAND DESCRIPTIONS

The remainder of this section contains descriptions of the Multics FAST subsystem commands, presented in alphabetical order. Each description contains the name of the command (including the abbreviated form, if any), discusses the purpose of the command, and shows the correct usage. Notes and examples are included when deemed necessary for clarity. The discussion below briefly describes the content of the various divisions of the command descriptions.

Name

The "Name" heading lists the full command name and its abbreviated form. The name is usually followed by a discussion of the purpose and function of the command and the expected results from the invocation.

Usage

This part of the command description first shows a single line that demonstrates the proper format to use when invoking the command and then explains each element in the line. The single line contains the full command name followed by the valid arguments. Some commands have required arguments; some commands have optional arguments. Most commands have both required and optional arguments; in general, the required arguments precede the optional arguments.

Any command argument preceded and followed by a brace ({}) is an optional argument. Any other argument is a required argument. Anything specifically identified as "control_arg" in the usage line must be preceded by a minus sign (-) in the actual invocation of the command. For example, the usage line:

```
commandname path {-control_arg} {xxx}
```

means that the command has one required argument and two optional arguments. Therefore, any of the following command lines are valid:

```
commandname path
commandname path -control_arg
commandname path xxx
commandname path -control_arg xxx
```

If a command accepts more than one of a specific type of argument, an "s" is added to the argument name. For example, the usage line:

```
commandname paths {-control_args}
```

means that the user must specify at least one pathname and may specify none, one, or several control arguments.

If a command accepts multiple arguments that must be in a specific order, the usage line is as follows:

commandname xxx1 yyy1 ... xxxn yyyn

to show that although several xxx and yyy arguments can be given, they must be given in pairs.

Notes

Comments or clarifications that relate to the command as a whole are given under the "Notes" heading. Also, where applicable, the required access modes, default condition (invoking the command without any arguments), and any special case information are included.

Examples

The examples show different valid invocations of the command. The results of each example command line are either shown or explained. Where input is to be typed in by the user, the line to be typed is preceded by an exclamation point (!), which is not to be typed. This convention is followed in all examples in this section.

Other Headings

Additional headings are used in some descriptions, particularly the more lengthy ones, to introduce specific subject matter. These additional headings may appear in place of, or in addition to, the notes.

Name: add_line_numbers, aln

The add_line_numbers command adds a new set of line numbers to a segment.

Usage

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

where:

1. path
   is the pathname of the segment to be modified.

2. new_number
   is the first line number to be added. If this argument is not specified, the default first line number is 100.

3. increment
   is the increment used to derive subsequent line numbers (10 by default). This argument can only be specified if the new_number argument is given.

Notes

The value of the new_number argument is used for the first line and the increment is added to derive subsequent numbers. If the text already has line numbers, these are retained but become part of the text on the line.

If the segment does not end with the newline character, the segment is truncated to the previous newline character.

Example

```
!   print_text data_1

    data_1  01/12/76  1119 mst Mon

    nonnumbered
    data segment
    input
    r 1120

!   add_line_numbers data_1          /default values are used
    r 1121

!   print_text date_1 -nhe           /-nhe suppresses the printing
    100 nonnumbered                  /of the header line
    110 data segment
    120 input
    r 1122
```

add_line_numbers

```
!     add_line_numbers data_1 500 5
      r 1122

!     print_text data_1  -nhe
      500 100 nonnumbered
      505 110 data segment
      510 120 input
      r 1123
```

Name: add_name, an

The add_name command adds alternate name(s) to the existing name(s) of a segment, multisegment file, directory, or link. See also the descriptions of the delete_name and rename commands.

Usage

add_name path names

where:

1. path
   is the pathname of a segment, multisegment file, directory, or link.

2. names
   are additional names to be added.

Notes

The user must have modify permission on the directory that contains the entry receiving the additional name.

The star and equal conventions can be used (see Section 1).

Two entries in a directory cannot have the same entryname; therefore, special action is taken by this command if the added name already exists in the directory that contains the path argument. If the added name is an alternate name of another entry, the name is removed from this entry, added to the entry specified by path, and the user is informed of this action. If the added name is the only name of another entry, the user is asked if he wishes to delete the other entry. If he answers "yes", the entry is deleted and the name is added to the entry specified by path; if he answers "no", no action is taken.

Examples

The command line:

```
add_name >my_dir>example.pl1 sample.pl1
```

adds the name sample.pl1 to the segment example.pl1 in the directory >my_dir.

add_name

The command line:

```
add_name >udd>**.private ==.public
```

adds to every entry having a name with private as the last component a similar name with public, rather than private, as the last component.

Name: basic

The basic command invokes the BASIC compiler to translate a segment containing BASIC source code. The object segment is saved in the user's working directory. Users of BASIC should refer to the Multics BASIC Manual, Order No. AM82.

Usage

```
basic path
```

where path is the pathname of the source program. The suffix of basic need not be given as part of the path argument. However, the basic suffix must be the last component of the name of the segment.

Examples

```
!    basic mpg.basic
     r 1245

!    basic test          /suffix of basic is assumed
     r 1246
```

Name: change, c

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

Usage

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

where:

1. /
   is a delimiter that can be any character that is not found in old_string or new_string except blank, tab, or digit.

2. old_string
   is a string of characters to be replaced.

3. new_string
   is a string of characters to be substituted for each occurrence of the old_string argument.

4. first_line
   is the line number of the first line to be changed.

5. last_line
   is the line number of the last line to be changed; if this argument is not given, the change is made only to the line specified by the first_line argument.

Note

All lines between and including the first and last lines specified are used. The line numbers specified by the first_line and last_line arguments do not have to appear in the text, but the range specified by them must contain at least one line.

Examples

```
!    130 for n = 1 to 5
!    140 let e = 40
!    150 for m = 1 to 3
!    160 let e = e + p(m)
!    change /e/s/ 150 170      /lines between and including line numbers 150
     r 1326                    /and 170 are used

!    print_text -nhe
!    130 for n = 1 to 5
!    140 let e = 40
```

change

```
!     150 for m = 1 to 3
!     160 lst s = s + p(m)      /"e" in "let" was changed
      r 1327

!     change  /lst/let/ 160
      r 1327

!     print_text  160
      160 let s = s + p(m)
      r 1328
```

Name: copy, cp

The copy command causes copies of specified segments and multisegment files to be created in the specified directories with the specified names. Access control lists (ACLs) and multiple names are optionally copied.

Usage

copy path1_1_ {path2_1_ ... path1_n_ path2_n_} {-control_args}

where:

1. path1_i_
   is the pathname of a segment or multisegment file to be copied. If path1 is the name of a link, the command copies the target of the link.

2. path2_i_
   is the pathname of a copy to be created from path1_i_. If the last path2 argument is not given, the copy is placed in the working directory with the entryname of path1_n_.

3. control_args
   can be chosen from the following list of control arguments:

   -name, -nm
   copies multiple names.

   -acl
   copies the ACL.

   -all, -a
   copies multiple names and the ACL.

   -brief, -bf
   suppresses the warning messages "Bit count inconsistent with current length..." and "Current length is not the same as records used...".

The control arguments can appear once anywhere in the copy command line after the command name and apply to the entire copy command line.

Notes

Read access is required for path1_i_. Status permission is required for the directory containing path1_i_. Append permission is required for the directory containing path2_i_. Modify permission is required if the -name, -acl, or -all control argument is used.

The star and equal conventions can be used (see Section 1).

If the ACL of a segment or multisegment file is being copied, then the initial ACL of the target directory has no effect on the ACL of the segment or multisegment file after it has been copied into that directory. The ACL remains exactly as it was in the original directory.

Since two entries in a directory cannot have the same entryname, special action is taken by this command if the name of the segment or multisegment file being copied (specified by path1_i) already exists in the directory specified by path2_i. If the existing entry has an alternate name, the entryname that would have resulted in a duplicate name is removed and the user is informed of this action; the copying operation then takes place. If the existing entry has only one entryname, the entry that already exists in the directory must be deleted to remove the name. The user is asked if the deletion should be done; if the user answers "no", the copying operation does not take place.

The copy command prints a warning message if the bit count of path1_i is less than its current length or if the current length is greater than the number of records used. These warnings are suppressed by the use of the -brief control argument.

Example

The command line:

```
copy >old_dir>fred.list george.=
```

copies segment or multisegment file named fred.list in the directory >old_dir into the working directory as george.list.

Name: delete, dl

The delete command causes the specified segments and/or multisegment files to be deleted.

Usage

delete paths {-control_args}

where:

1. paths
   are the pathnames of segments or multisegment files to be deleted. The star convention can be used.

2. control_args
   can be chosen from the following:

   -brief, -bf
   inhibits the printing of an error message if a segment or multisegment file to be deleted is not found.

   -force
   deletes the specified entries whether or not they are protected, without issuing a query.

Notes

In order to delete a segment or multisegment file with the delete command, the entry must have both its safety switch and its copy switch off and the user must have modify permission on the containing directory. If either switch is on, the user is interrogated as to whether he wishes to delete the entry.

If any one of the paths is a link, delete prints a message and does not delete either the path in question or the link. (See the description of the unlink command.) If any one of the paths is a directory, delete prints a message; it does not delete the directory.

Name: delete_acl, da

The delete_acl command removes entries from the access control lists (ACLs) of segments, multisegment files, and directories. For a description of ACLs, see Section 4.

Usage

delete_acl {path} {User_ids} {-control_args}

where:

1. path
   is the pathname of a segment, multisegment file, or directory. If it is -wd, -working_dir, or omitted, the working directory is assumed. If path is omitted, no User_id 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 deleted. (For a description of the matching strategy, refer to the set_acl command.) If no User_id is given, the user's Person_id and current Project_id are assumed.

3. control_args
   can be chosen from the following:

   -all, -a
   deletes the entire ACL with the exception of an entry for *.SysDaemon.*.

   -directory, -dr
   deletes ACLs for only directories. The default is segments, multisegment files, and directories.

   -segment, -sm
   deletes ACLs for only segments and multisegment files.

   -brief, -bf
   suppresses the message "User name not on ACL."

Notes

If the delete_acl command is invoked with no arguments, it deletes the entry for the user's Person_id and current Project_id on the ACL of the working directory.

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

The user needs modify permission on the containing directory.

Examples

The command line:

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

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

The command line:

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

deletes from the ACL of every segment, multisegment file, and directory (in the working directory) whose entryname has a first component of beta all entries except the one for *.SysDaemon.*.

The command line:

```
da beta.** .. -sm
```

deletes from the ACL of only all segments and multisegment files (in the working directory) whose entryname has a first component of beta all entries except the one for *.SysDaemon.*.

delete_line_numbers

Name: delete_line_numbers, dln

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

Usage

```
delete_line_numbers path
```

where path is the pathname of the segment to be modified.

Note

Unnumbered text can be created and modified only by using the edm command. New line numbers can be added to unnumbered text with the add_line_numbers command.

Example

```
!    print_text  data

     data  01/12/76  1543.1 mst Mon

!    10 ten
!    20 twenty
!    30  thirty
     r 1543

!    delete_line_numbers data
     r 1544

!    print_text data

     data  01/12/76  1544.7 mst Mon

     ten
     twenty
     thirty
     r 1545
```

Name: delete_name, dn

The delete_name command deletes specified names from segments, multisegment files, links, or directories that have multiple names. See the descriptions of the add_name and rename commands for adding and changing names, respectively.

Usage

delete_name paths

where paths are the pathnames that are to be deleted.

Notes

In keeping with standard practice, each path can be a relative pathname or an absolute pathname; its final portion (the storage system entryname in question) is deleted from the segment or directory it specifies, provided that doing so does not leave the segment or directory without a name. In this case, the user is interrogated as to whether or not he wishes the segment or directory in question to be deleted.

The user must have modify permission on the containing directory.

The star convention can be used. For a description of the star convention, see Section 1.

Example

The command line:

```
delete_name alpha >my_dir>beta
```

deletes the name alpha from the list of names for the appropriate entry in the current working directory and also deletes the name beta from the list of names for the appropriate entry in the directory >my_dir. Neither alpha nor beta can be the only name for their respective entries.

Name: delete_text, dt

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

Usage

```
delete_text first_line {last_line}
```

where:

1. first_line
   is the line number of the first line to be deleted.

2. last_line
   is the line number of the last line to be deleted. If this number is not given, only the line specified by the first_line argument is deleted.

Note

All lines between and including the first and last lines specified are deleted. The line numbers specified by the first_line and last_line arguments need not appear in the text, but the range specified by them must contain at least one line.

Example

```
!    print_text eval.basic

     eval.basic  01/16/76  1023 mst Fri

     100 input n
     110 for i = 1 to n
     120 input x
     130 let t1 = t1 + x
     140 let t2 = t2 + y
     150 if t2 > 9000 then 500
     160 next i
     r 1024

!    delete_text 140 155      /deletes all lines between and including lines
     r 1025                   /140 and 155

!    print_text -nhe
     100 input n
     110 for i = 1 to n
     120 input x
     130 let t1 = t1 + x
     160 next i
     r 1025
```

Name: dprint, dp

The dprint (daemon print) command queues specified segments and/or multisegment files for printing on one of the Multics line printers. The output is by default identified by the requestor's Person_id. This command does not work on standard object segments.

Usage

dprint {-control_args} {paths}

where:

1. control_args
   can be chosen from the following:

   -access_label, -albl
   for each pathi specified, uses the access class of that segment as a label at the top and bottom of every page (see "Notes" below).

   -brief, -bf
   suppresses the message "j requests signalled, k already queued. (request_type queue)." This control argument cannot be overruled later in the command line. (See the -request_type and -queue control arguments below.)

   -bottom_label STR, -blbl STR
   uses the specified string as a label at the bottom of every page (see "Notes" below).

   -copy N, -cp N
   prints N copies ($N \leq 4$) of specified paths. This control argument can be overruled by a subsequent -copy control argument. If pathi is to be deleted after printing, all N copies are printed first. If this control argument is not given, one copy is made.

   -delete, -dl
   deletes (after printing) specified paths.

   -destination STR, -ds STR
   labels subsequent output with the string STR, 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.

   -header STR, -he STR
   identifies subsequent output by the string STR. 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.

   -indent N, -in N
   prints so that the left margin is indented N columns. If this control argument is not given, no indentation occurs.

-label STR, -lbl STR
uses the specified string as a label at the top and bottom of every page (see "Notes" below).

-line_length N, -ll N
prints so that lines longer than N characters are continued on the following line, i.e., no line of output extends past column N. If this control argument is not given, a line length of 136 characters is used.

-no_endpage, -nep
prints so that the printer skips to the top of a page only when a form-feed character is encountered in the input path. This argument causes the -page_length control argument, if present, to be ignored.

-no_label, -nlbl
does not place any labels on the printed output.

-non_edited, -ned
prints nonprintable control characters as octal escapes rather than suppressing their printing.

-notify, -nt
sends a confirming message when the requested output is done, showing the pathname and charge.

-page_length N, -pl N
prints so that no more than N lines are on a page. If this control argument is not given, a page length of 60 lines is used.

-queue N, -q N
prints in priority queue N ($N \leq 3$). This control argument can be overruled by a subsequent -queue control argument. If this control argument is not given, queue 3 is assumed. (See "Notes" below.)

-request_type STR, -rqt STR
places dprint requests specified paths in the queue for requests of the type identified by the string STR (see "Notes" below). If this control argument is not given, the default request type is "printer".

-single, -sg
prints so that any form-feed or vertical-tab character is printed as a single newline character.

-top_label STR, -tlbl STR
uses the specified string as a label at the top of every page (see "Notes" below).

-truncate, -tc
prints so that any line exceeding the line length is truncated rather than "folded" onto subsequent lines.

2. paths
are the pathnames of segments to be queued for printing.

Notes

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

If control arguments are present, they affect only paths following them in the command line. If control arguments are given without a following path argument, they are ignored for this invocation of the command and a warning message is returned.

The -queue 1 control argument places requests in the top priority queue, -queue 2 places them in the second priority queue, -queue 3 (or not specifying a queue) places them in the third priority queue, and -queue 4 places them in the lowest priority queue. All requests in the first queue are processed before any requests in the other queues, and so on. Higher priority queues usually have a higher cost associated with them.

The -brief, -delete, -single, -truncate, and -no_endpage control arguments cannot be reset in a given invocation of the command; e.g., once -delete appears in a line, all subsequently specified paths are deleted after printing.

The -request_type control argument is used to ensure that a request is performed by a member of a particular group of printers, e.g., to distinguish between onsite printers and remote printers at various locations, or between printers being charged to different projects. Only request types of generic type "printer" can be specified. Request types can be listed by the print_request_types command.

If a requested output operation cannot be done, the daemon process sends a message to the user of the form:

Request path reason.

The -label, -top_label, -bottom_label, and -access_label control arguments allow the user to place labels on each page of printed output. The default labels are access labels, i.e. the -access_label control argument is assumed. These control arguments are read, in sequence, from left to right by the dprint command. For example, if -access_label is specified, it is printed at the top and bottom of the page. If the next control argument is -top_label STR, then the top access label becomes STR but the bottom label remains the same. Each label control argument can override the preceding one. The label lines are printed on the second line of the page and on the next to last line of the page. Note that if the access class of pathi is system_low and the access class name defined for system_low is null, then the default access label is blank. The default access label can be overridden by the -no_label control argument if labels are not wanted or by one of the other label-related control arguments.

The top and bottom labels are treated independently. Thus, use of the -top_label control argument alone leaves an access label as the default bottom label. A page label that exceeds 136 characters is truncated to that length. Only the first line of a page label is printed, i.e., a new line terminates the page label. Form feeds and vertical tabs are not permitted. The various label control arguments are incompatible with the -no_endpage control argument and they are ignored independent of the position in the command line of the -no_endpage control argument.

Paths cannot be printed unless appropriate system processes have sufficient access. The process that runs devices of the specified class (normally IO.SysDaemon) must have read access to all paths to be printed and status permission on the containing directory. Pathi cannot be deleted after printing unless its safety switch is off and the system process has at least sm access on the containing directory. Also, pathi is not deleted if it has a date-time-modified value later than the date-time-modified value at the time of the dprint request.

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

Example

The command line:

```
dp -he Jones -cp 2 -dl test1 test7 -he Doe -ds BIN-5 text.runout
```

causes two copies of each of the segments named test1 and test7 in the current working directory to be printed with the header "Jones" and then deleted. It also causes two copies of the segment named text.runout in the current working directory to be printed with the header "Doe" and destination "BIN-5", then deleted.

Name: edm

The edm command invokes a simple Multics context editor. It is used for creating and editing ASCII segments. This command cannot be called recursively. See the Multics User's Guide (Order No. AL40) for an introduction to the use of edm.

Usage

edm {path}

where path specifies the pathname of the segment to be created or edited. The path argument can be either an absolute or a relative pathname. If path is not given, edm begins in input mode (see "Notes" below), ready to accept whatever is subsequently typed as input. If path is given, but the segment does not yet exist, edm also begins in input mode. If path specifies a segment that already exists, edm begins in edit mode.

Notes

This command operates in response to requests from the user. To issue a request, the user must cause edm to be in edit mode. This mode is entered in two ways: if the segment already exists, it is entered automatically when edm is invoked; if dealing with a new segment (and edm has been in input mode), the mode change character must be issued. The mode change character is the period (.), issued as the only character on a line. The command announces its mode by typing "Edit." or "Input." when the mode is entered. From edit mode, input mode is also entered via the mode change character.

The edm requests are predicated on the assumption that the segment consists of a series of lines to which there is a conceptual pointer that indicates the current line. (The "top" and "bottom" lines of the segment are also meaningful.) Various requests explicitly or implicitly cause the pointer to be moved; other requests manipulate the line currently pointed to. Most requests are indicated by a single character, generally the first letter of the name of the request. For these requests only the single character (and not the full request name) is accepted by the command. Certain requests have been considered sufficiently dangerous, or likely to confuse the unwary user, that their names must be specified in full.

If the user issues a quit signal while in edit mode and then invokes the program_interrupt command, the effect of the last request executed on the edited copy is nullified. (See the description of the program_interrupt command in this document.) In addition, any requests not yet executed are lost. If program_interrupt is typed after a quit in comment or input modes, then all input since last leaving edit mode is lost. If the user wishes to keep the input, he must invoke the start command following the quit.

In the examples that follow, the pointer that indicates the current line is represented by an arrow (->).

Requests

The requests are as follows (detailed descriptions follow the list, in the same order):

-	backup
=	print current line number
,	comment mode
.	mode change
b	bottom
c	change
d	delete
E	execute
f	find
i	insert
k	kill
l	locate
merge	insert segment
move	move lines within segment
n	next
p	print
q	quit
qf	quitforce
r	retype
s	substitute
t	top
updelete	delete to pointer
upwrite	write to pointer (upper portion of segment)
v	verbose
w	write

Backup (-)

Format: - n

Purpose: Move pointer backwards (toward the top of the segment) the number of lines specified by the integer n.

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

Pointer: Set to the nth line specified before the current line.

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

Example: Before:

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

Request: -2

After:

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

Print Current Line Number (=)

Format: =

Purpose: Print current line number.

Pointer: Unchanged.

Comment Mode (,)

Format: ,

Purpose: Establish a special inputting mode in which the lines, starting with the current one, are successively treated as follows. The line is printed without a carriage return and anything then typed by the user (e.g., comment, newline, etc.) is appended to the line. If the user types the mode change character (".") as his comment, the last line typed is unchanged and edit mode is reentered.

Pointer: Left pointing to the last line printed.

Mode Change (.)

Format: .

Purpose: Allow user to enter edit mode from input mode or vice versa. This request is also used to terminate the comment mode request and return edm to edit mode.

Pointer: Left pointing to the last line input, edited, or printed.

Bottom (b)

Format: b

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

Pointer: Set after the last line in the segment.

Change (c)

Format: c n /string1/string2/

Purpose: Replace every instance of string1 by string2 in the n consecutive lines beginning with the current line, where n must be an integer. If the user is in verbose mode, edm prints each line that is changed (see the v request). If no line is changed, then edm prints "edm: Substitution failed."

Spacing: A space before n and between n and the string1 delimiter is optional.

edm

Pointer: Set to the last line scanned.

Delimiters: Any character not appearing in string1 or string2 can delimit the strings (/ is used as the delimiter in the format line). A delimiter following string2 is optional.

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

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

Request: c2/./;/

Response:
```
x = y;
q = r;
```

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

Note: For compatibility with qedx, this request can also be given as s (for substitute).

Delete (d)

Format: d n

Purpose: Cause n lines to be deleted where n is an integer. Deletion begins at the current line.

Spacing: A space is optional between d and n.

Pointer: Set to "no line" following the line deleted. That is, an i (insert) request or a change to input mode would take effect before the next nondeleted line.

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

Note: The requests c, d, n, and p count "no line" when issued immediately after a delete request.

Execute (E)

Format: E command_line

Purpose: Pass commandline to the command processor for execution as a command line.

Spacing: A single space following E is not significant.

Pointer: Unchanged.

Find (f)

Format: f string

Purpose: Search segment for a line beginning with string. Search starts at the line following the current line and continues around the entire segment until the string is found or until the pointer returns to the current line. The current line is not searched. If the string is not found, the error message "edm: Search failed." is printed. If the string is found and the user is in verbose mode, the line containing the string is printed.

Spacing: A single space following f is not significant. All other leading and embedded spaces are used in searching.

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

Default: If the string is null, the string used by the last f or l (locate) request is used.

Insert (i)

Format: i newtextline

Purpose: Insert newtextline after the current line.

Spacing: The first space following i is not significant. All other leading and embedded spaces become part of the text of the newtextline.

Pointer: Set to the inserted line.

Default: If newtextline is null, a blank line is inserted.

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

Kill (k)

Format: k

Purpose: To inhibit (kill) responses following a c, f, l, n, or s request.

Pointer: Unchanged.

Note: See v (verbose) request for restoring responses.

Locate (l)

Format: l string

Purpose: Search segment for a line containing string. Search starts at the line following the current line and continues around the entire segment until the string is found or until the pointer returns to the current line. The current line is not searched. If the string is not found, the error message "edm: Search failed." is printed. If the string is found and the user is in verbose mode, the line containing the string is printed.

Spacing: A single space following l is not significant. All other leading and embedded spaces are used in searching.

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

Default: If the string is null, the string used by the last l or f (find) request is used.

Example: Before:

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

Request: `l = y`

Response: `x = y;`

After:

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

Merge (merge)

Format: merge path

Purpose: Insert the segment specified by the pathname path after the current line. The pathname specified by path can be either an absolute or a relative pathname.

Spacing: A single space following merge is not significant.

Pointer: Set to "no line" following the last line of the inserted segment.

Default: If path is not given, the pathname given in the invocation of edm is used. If a pathname is given neither in this request nor in the invocation of edm, an error message is printed and edm looks for another request.

Move (move)

Format: move m n

Purpose: Insert n lines beginning at line m after the current line and delete them from their original location.

Spacing: A space is optional before m.

Pointer: Set to "no line" following the lines moved. That is, an i request or change to input mode would take effect immediately following the moved text.

Default: If n is null, only the single line m is moved.

Note: The requests c, d, n, and p count "no line" when issued immediately after a move request.

Next (n)

Format: n n

Purpose: Move pointer down the segment n lines. The line so located is printed if the user is in verbose mode.

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

Pointer: Set to the nth line specified after the current line.

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

Note: The printed response to this request can be shut off using the k (kill) request.

Print (p)

Format: p n

Purpose: Print n lines beginning with the current line.

Spacing: A space is optional between p and the integer n.

Pointer: Left pointing to the last line printed.

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

Note: A print request in edm can be aborted by issuing a quit signal and typing pi or program_interrupt. This puts edm in a state where it is ready to accept another request. (See the description of the program_interrupt command.)

Quit (q)

Format: q

Purpose: Exit edm and return to the caller, usually command level. If no write request has been made since the last change to the edited text, edm warns the user that the changes will be lost and asks if he still wishes to quit.

Pointer: If the user is queried and answers no, then the pointer is unchanged.

Quitforce (qf)

Format: qf

Purpose: Exit from edm directly without either warning or querying the user.

Retype (r)

Format: r newtextline

Purpose: Replace current line with newtextline.

Spacing: One space between r and newline is not significant. All other leading and embedded spaces become part of the text of the newtextline.

Pointer: Unchanged.

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

Substitute (s)

Note: This request is identical to the c (change) request.

Top (t)

Format: t

Purpose: Move pointer to the top of the segment.

Pointer: At "no line" immediately above the first line of text.

Note: An i (insert) request immediately following a t request causes insertion of a text line at the beginning of the segment.

Delete to Pointer (updelete)

Format: updelete

Purpose: Delete all lines above (but not including) the current line.

Pointer: Unchanged.

Write to Pointer (upwrite)

Format: upwrite path

Purpose: Save all the lines above the current line (but not including the current line) in the segment whose name is specified by path. The lines written out are deleted from the edit buffers and thus are no longer available for editing. They will replace the previous contents of path. The pathname specified by path can be either an absolute or a relative pathname.

Spacing: A single space following upwrite is not significant.

Pointer: Unchanged.

Default: If path is not given, the pathname given in the invocation of edm is used. If a pathname is given neither in this request nor in the invocation of edm, an error message is printed and edm looks for another request.

Verbose (v)

Format: v

Purpose: Cause edm to print responses following a c, f, l, n, or s request. This is the default mode.

Pointer: Unchanged.

Note: See k (kill) for inhibiting verbose mode.

Write (w)

Format: w path

Purpose: Write out (save) the edited copy in the segment specified by path. The pathname specified by path can be either an absolute or a relative pathname.

Spacing: A space between w and path is not significant.

Pointer: If the w request is successful, set to "no line" at the end of the segment.

Default: If path is not given, the pathname given in the invocation of edm is used. If a pathname is given neither in this request nor in the invocation of edm, an error message is printed and edm looks for another request.

Note: To terminate editing without saving the edited copy, see the qf (quitforce) request.

Names: enter, e
enterp, ep

These prelogin requests are used by anonymous users to gain access to Multics. Either one is actually a request to the answering service to create a process for the anonymous user.

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}
enterp {anonymous_name} Project_id {-control_args}
```

where:

1. anonymous_name
   is an optional identifier that is not checked by the system, but is passed to the user's process overseer as if it were a person identifier. If anonymous_name is not specified, it is assumed to be the same as the project identifier.

2. Project_id
   is the identification of the user's project.

3. control_args
   can be selected from the following:

   -brief, -bf
   suppress messages associated with a successful login. If the standard process overseer is being used, then the message of the day is not printed.

   -force
   log the user in if at all possible, provided the user has the guaranteed login attribute. Only system users who perform emergency repair functions have the necessary attribute.

-home_dir path, -hd path
set the user's home directory to the path specified, if the user's project administrator allows him to specify his home directory.

-modes STR, -mode STR, -md STR
set the I/O modes associated with the user's terminal to STR, where the string STR consists of modes acceptable to the tty_ I/O module. The STR string is usually a list of modes separated by commas; the STR string must not contain blanks. (See "Examples" below.)

-no_preempt, -np
refuse to log the user in if he can only be logged in by preempting some other user in his load control group.

-no_print_off, -npf
cause the system to overtype a string of characters to provide a black area for the user to type his password.

-no_start_up, -ns
instruct the standard process overseer not to execute the user's start_up.ec segment, if he has one, and if the project administrator allows him to avoid it.

-no_warning, -nw
suppress even urgent system warning and emergency messages from the operator, both at login and during the user's session. Use of this argument is recommended only for users who are using a remote computer to simulate a terminal, or are typing out long memoranda, when the process output should not be interrupted by even the most serious messages.

-outer_module p, -om p
attach the user's terminal via the outer module named p rather than the user's registered outer module, if the user has the privilege of specifying his outer module.

-print_off, -pf
suppress overtyping for the password.

-process_overseer path, -po path
set the user's process overseer to the procedure given by the path specified, if the user's project administrator allows him to specify his process overseer. If path ends in the characters ",direct", the specified procedure is called directly during process initialization rather than by the init_admin procedure provided by the system. This means that the program specified by path must perform the tasks that would have been performed by the init_admin procedure.

-ring N, -rg N
set the user's initial ring to be ring N, if this ring number is greater than or equal to the user's registered initial ring and less than his registered maximum ring.

-subsystem path, -ss path
create the user's process using the prelinked subsystem in the directory specified by path. The permission to specify a process overseer, which may be given by the user's project administrator, also governs the use of the -subsystem argument. To override a default subsystem by the project administrator, type -ss "".

-terminal_type STR, -ttp STR
set the user's terminal type to STR, where STR is any terminal type name defined in the standard terminal type table. (To obtain a list of terminal types, refer to the print_terminal_types command.) This control argument overrides the default terminal type.

Notes

If neither the -print_off nor -no_print_off control argument is specified at log-in, the system attempts to choose the option most appropriate for the user's terminal type.

If the project administrator does not allow the user to specify the -subsystem, -outer_module, -home_dir, -process_overseer, or -ring control arguments or if he does allow one or more of these control arguments and they are incorrectly specified by the user, a message is printed and the login is refused.

fortran

Name: fortran, ft

The fortran command invokes the FORTRAN compiler to translate a segment containing the text of a FORTRAN source program into an object segment. The object segment is saved in the user's working directory. Users of FORTRAN should refer to the Multics FORTRAN Reference Manual (Order No. AT58), and the Multics FORTRAN Users' Guide (Order No. CC70).

Usage

fortran path {control_arg}

where:

1. path
   is the pathname of the source program. The suffix of fortran need not be given as part of the path argument. However, the fortran suffix must be the last component of the name of the segment.

2. control_arg
   is -no_line_numbers (or -nln) to specify that the source segment contains text without line numbers.

Examples

```
! fortran mpg.fortran
  r 1510

! fortran test                    /a suffix of fortran is assumed
  r 1511
```

help

Name: help

The help command prints online information about a specific topic (an info). Information is maintained online about FAST subsystem commands and some general topics. A list of topics available is printed by issuing the command with "topics" as its argument.

Usage

help {info_names}

where info_names are names of specific topics, such as command names.

help

Pages 42 through 54 of this section have been deleted.

and gives a description of the -brief control argument of the mail command. However, help reports an error for the command:

```
help mail -ca -brief -match -exclude
```

because -match and -exclude are treated as control arguments to the help command, rather than as selection operands to -ca. Note that, if the command:

```
help mail -ca -header -brief
```

is given, help does not report an error. The -brief causes help to print a summary of the mail command. The -ca -header then prints a description of mail's -header control argument. In this case, -brief is accepted by help as a control argument, so no error is reported.

-all, -a
prints the entire info or subroutine entry point description without asking the user questions.

STARTING PARAGRAPH

Normally, help begins printing the first paragraph in the info. The control arguments below can select a particular section and/or a particular paragraph at which printing is to start.

-section STRs, -scn STRs
begins printing the section whose title contains the strings STRs. The entire section title is not required. Instead, the first section whose title contains all of the strings STRs is selected. The strings can appear in the section title in any order. The strings can be typed in lowercase, since case is ignored during matching operations. All arguments following the -section control argument until the next control argument are treated as STRs.

-search STRs, -srh STRs
begins printing with the first paragraph containing strings STRs. All of the strings must appear in the selected paragraph, but they can appear in any order. The strings can be typed in lowercase, since case is ignored when matching. All arguments following the -search control argument are treated as STRs, so -search must be the last control argument. The search usually begins with the first paragraph, but when -section is also given it begins with the matching section and continues to the last paragraph (i.e., without wraparound).

When -section or -search control arguments are given and no matching paragraph is found in one of the infos selected by an info_name or info selection control argument, then that info is skipped without comment. Thus, the starting paragraph control arguments serve as a secondary info selection mechanism.

The starting paragraph control arguments can be used with any of the information selection control arguments listed above, but its effect differs depending upon which of them are used. When -section or -search are used with -header, only the heading lines for infos containing a matching paragraph are listed. The matching paragraph itself is not printed. When they are used with -brief or -control_arg, help prints a heading line and then the information selected by -brief or -control_arg. The matching paragraph is not printed.

When -section or -search arguments are used with -no_header, a brief heading line is printed preceding the matching paragraph. When used with -title, help prints a heading line, then the list of section titles, and finally the matching paragraph. When used with -all, the entire info is printed for infos containing a matching paragraph.

PARAGRAPH GROUPING

The following control arguments determine how much information help prints before asking the user if he wants to see more.

-minlines I
sets the minimum paragraph size to I lines. Paragraphs smaller than this size are printed with preceding paragraphs. The default is 4.

-maxlines J
sets the maximum paragraph grouping size to J lines. When paragraphs are grouped together, the number of grouped lines may not exceed this size. The default is 15.

For example, consider an info divided into paragraphs as follows:

```
Paragraph 1 (8 lines)
 (2 blank lines)
Paragraph 2 (3 lines)
 (2 blank lines)
Paragraph 3 (4 lines)
```

With -minlines 4 and -maxlines 15, help treats paragraph 2 as a short paragraph which is printed with paragraph 1 (total lines = 13). However, paragraph 3 is 4 lines long, and is treated as a distinct paragraph.

With -minlines 5 and -maxlines 10, help prints paragraph 1 separately, since grouping short paragraph 2 with paragraph 1 would print 13 lines, exceeding -maxlines. Paragraphs 2 and 3 are grouped together (total lines = 9) because paragraph 3 is shorter than 5 lines.

Paragraphs that have been seen are not grouped with unseen paragraphs. Similarly, paragraphs at the end of one section of info are not grouped with those beginning another section. Paragraphs are not grouped when -section or -search control arguments are used to find a particular starting paragraph. If the wrong paragraph is found by the search, grouping could compound the error by printing more of the wrong information. For similar reasons, grouping is suppressed when the section and search responses are used.

Responses to Questions Asked by the help Command

The responses accepted when help questions the user are given in the list below. Those responses that search the info or list section titles operate from the current paragraph to the end of the info. No wraparound feature is employed. However, -top or -t can be used with these responses to cause searching or listing from the top of the info, rather than from the current paragraph.

The help command remembers which paragraphs the user has seen and which have been skipped or not yet reached. It asks the user to "Review" paragraphs seen before, but asks whether "More help" is needed for unseen paragraphs. It stops printing if all paragraphs have been seen when the end of info is reached. However, if any paragraphs were skipped, help asks whether the user wants to see them. If the response is "yes", the first unseen paragraph is printed. The user can then answer "skip -seen" to view subsequent unseen paragraphs.

The responses to all questions asked by the help command can be chosen from the following:

brief, bf
prints a summary of a command, active function or subroutine info, including Syntax section and a list of control arguments. The help command then repeats the previous question.

control_arg STRs, ca STRs
prints descriptions of control (or other) arguments whose names contain one of the strings STRs. The help command then repeats the previous question.

entry_point {EP_NAME}, ep {EP_NAME}
skips to the description of subroutine entry point EP_NAME. The EP_NAME can be given as entry_point_name or subroutine_$entry_point_name. For example:

```
ep rsnnl
```

when in the info segment describing the ioa_ subroutine, skips to the description of the ioa_$rsnnl entry point. If EP_NAME is omitted, help skips to the description of the subroutine_$subroutine_ entry point.

header, he
prints a long heading line to identify the current info. The line includes: pathname of the info, info heading, and line count.

no, n
exits from the current info, and begins printing the next info selected by info_names given in the help command. Returns from the help command if all selected infos have been printed.

quit, q
causes the help command to return without printing the remaining infos selected by the info_names.

rest {-scn}, r {-scn}
prints the rest of the info without intervening questions. If -section or -scn control arguments are given, then help prints only the rest of the current section without questions. When the section has been printed, help then asks the user if he wants to see the next section.

search {STRs} {-top}, srh {STRs} {-top}
skips to next paragraph containing all strings STRs. Paragraph selection is performed as described above for the -search control argument. If -top or -t is given, searching starts at the beginning of the info. If STRs are omitted, help uses the strings from the previous search response or -search control argument. If the search fails, help prints the message:

No matching paragraph found.

and repeats the previous question.

section {STRs} {-top}, scn {STRs} {-top}
skips to the next section whose title contains all strings STRs. Title matching is performed as described above for the -section control argument. If -top or -t is given, title searching starts at the beginning of the info. If STRs are omitted, help uses the search strings from the previous section response or -section control argument. If the search fails, help prints the message:

No matching section found.

and repeats the previous question.

skip {-scn} {-rest} {-seen} {-ep},
s {-scn} {-rest} {-seen} {-ep}
skips the next paragraph and asks the user if he wants to see the paragraph that follows it. If -section or -scn is given, help skips all paragraphs of the current section. If -rest or -r, -entry_point or -ep are given, help skips the rest of this info or subroutine entry point description, continuing with the next. If -seen is given, help skips to the next paragraph that the user has not seen. Only one of these control arguments can be used at a time.

title {-top}
lists titles and line counts of all sections remaining in the current info. If -top or -t is given, help lists all section titles.

top, t
skips to the beginning of the info, prints the heading line, and asks the user if he wants to see the first section.

yes, y
prints the next paragraph of information, then asks the user if he wants more help.

?
prints a list of available responses.

.
prints "help" to identify the current interactive environment.

.. command_line
passes the remainder of the response to the Multics command processor as a command line.

Search List

The help command uses the "info_segments" search list, which has synonyms of "info_segs" and "info". For more information about search lists, see the descriptions of the search facility commands and, in particular, the add_search_paths command description in this manual. Type:

```
print_search_paths info
```

to see what the current "info" search list is. The default search list is:

```
>doc>iml_info
>doc>info
```

Notes

When the star convention is used, the help command performs the following steps:

1. The info segments whose entryname matches any of the star names are alphabetized within their directory and scanned in that order.

2. When -section and -search control arguments are given, help scans the matching infos until the desired sections and/or paragraphs are found. If a matching paragraph is found, help prints it. Then help asks the user if he wants to see remaining paragraphs. Note that any section and search responses given at this point scan only the current info. If a matching paragraph is not found in one of the infos selected by star name, then that info is skipped without comment. Thus, it is possible to scan all info segments and print only those containing certain section titles or certain words.

3. When -section and -search control arguments are not given, help begins printing the first paragraph of each info that matches any of the starnames. Then help asks the user if he wants to see remaining paragraphs.

4. The -title, -all, -brief and -control_arg control arguments apply to each info selected by the star names and -section/-search string matching. Section titles, a brief summary or particular control argument descriptions are printed before the matching paragraph. When -all is given with -section or -search, the entire info selected by the string matching is printed without questions.

5. The yes, no, rest and skip responses operate on the next selected paragraph. This paragraph may be the first paragraph of the next selected info, or even the first paragraph that matches the -section and -search criteria in the next selected info.

6. If the user issues a quit signal, the program_interrupt command can be used to reenter the interactive help environment. The question asked previously is repeated.

Info Naming Conventions

Infos for Multics commands, active functions and subroutines are given the name of the particular system module with a suffix of info. For example, the info describing the pl1 compiler command is called:

pl1.info

Information about changes made to a command or active function from one release to the next are given the name of the particular system module with a suffix of changes.info. For example, changes to the fortran compiler are described in fortran.changes.info.

General information describing features or use of the system is included in infos whose names end with a suffix of gi.info (gi for general information). For example, acl_matching.gi.info describes how Access Control List entries are matched with User_ids in access control commands such as set_acl.

More than 500 infos are available. To find information about a particular area of the system, use the -header control argument with an entryname containing stars to list the names of available infos. For example, to list info_names related to the FORTRAN compiler, you could type:

```
help fortran*.** -he
```

To get a list of all general information segments, type:

```
help *.gi -he
```

Info Segment Format

Users can create info segments describing their own commands, exec_coms and application programs. Info segments must be formatted in a special way so that the help command can parse them into paragraphs. For information about this format, type:

```
help info_seg.gi
```

Examples

In the examples given below, the lines typed by the user are indicated by an exclamation point at the beginning of the line or immediately preceding a request. In the first example, the user wants to see list.info:

```
! help list
  (6 lines follow;  131 lines in info)
  04/10/77  list, ls

  Syntax:  ls {entrynames} {-control_args}


  Function:  prints information about the entries in a single directory.

  Arguments (9 lines).  More help?  ! yes

  Arguments:
  entrynames
     are the names of entries to be listed.  The star convention can be used.
     If no entrynames are given, all entries in the directory (of the default
     types or the types specified by control arguments) are listed.  A
     pathname can be given instead of an entryname, causing the entries
     specified by its entryname portion to be listed, in the directory
     specified by its directory portion.  It is an error to specify more than
     one directory to be listed in a single invocation of the list command.

  Control arguments (5 lines).  More help?  ! no
  r 1459 0.753 35.793 744
```

In the following example, the user knows what the "Syntax" and "Function" sections are but wants to see the control arguments section for the list command. Note that the argument for the -section control argument can be upper or lower case.

```
! help list -scn Control
  (7 lines follow;  131 lines in info)
  04/10/77  list, ls

  Control arguments:  described below according to their functions.
  DIRECTORY
  -pathname path, -pn path
     list entries in the directory named path.  Note the restriction
     described above under Arguments.

  15 more lines.  More help?  ! yes

  ENTRY TYPE
  -segment, -sm
     list segments
  -multisegment_file, -msf
     list multisegment files
  -file, -f
     list files (segments and multisegment files)
  -directory, -dr
     list directories
  -branch, -br
     list branches (segments, multisegment files, and directories)
  -link, -lk
     list links
  -all, -a
     list all four entry types

  12 more lines.  More help?  ! no
  r 1500 0.215 12.593 264
```

In the following example, the user is searching for all the list commands that have the word request anywhere in the info.

```
! help list_*.info -srh request
>doc>info>list_abs_requests.info  (6 lines follow;  56 in info)


07/20/78  list_abs_requests, lar
          list_daemon_requests, ldr
          list_retrieval_requests, lrr

Syntax:  (lar ldr lrr) {rel_path} {-control_args}

Function & Arguments (9 lines).  More help?  ! yes

Function:  these  commands list  requests in the  absentee, I/O daemon, and
retrieval queues, respectively.


Arguments:
rel_path
   is  relative  pathname of  request(s) to  be  listed.  It  can end in  a
   starname.  Default is  to list requests of all  pathnames.  See also the
   -entry control argument.

Control arguments (34 lines).  More help?  ! no

>doc>info>list_carry_requests.info  (3 lines follow;  22 in info)
07/12/78  list_carry_requests, lcr

Syntax:  lcr {-control_args}

Function & Control arguments (12 lines).  More help?  ! no
r 1732 2.585 40.663 647
```

If the user wishes to see the info for the help_ subroutine with all the entry points described, he can do the following:

```
! help help_
>doc>info>help_.info
(3 lines follow, 9 in introduction;  74 lines, 4 entry points in info)
12/11/78  help_

Function:  This subroutine performs the basic work of the help command.

Entry points in help_ (4 lines).  More help?  ! yes

Entry points in help_:
12/11/78  help_$init                12/11/78  help_$check_info_segs
12/11/78  help_ (entry point)       12/11/78  help_$term

Entry:  12/11/78  help_$init
(10 lines follow;  16 lines in entry point)  More help?  ! yes

Syntax:
     declare help_$init entry (char(*), char(*), char(*), fixed bin, ptr,
          fixed bin(35));

     call help_$init (caller, search_list_name, search_list_ref_dir,
          required_version, Phelp_args, code);
```

help

This page intentionally left blank.

how_many_users

Name: how_many_users, hmu

The how_many_users command tells how many users are currently logged in on the system. In addition, it prints the name of the system, the load on the system, and the maximum load. If the absentee facility is running, the number of absentee users and the maximum number of absentee users is printed also.

Usage

how_many_users {-control_args} {optional_args}

where:

1. control_args
   can be chosen from the following control arguments:

   -long, -lg
   prints additional information including the name of the installation, the time the system was brought up, the time of the next shutdown, if it has been scheduled, and the time of the last shutdown or crash. Load information on absentee users is also printed.

   -absentee, -as
   prints load information on absentee users only, even if the absentee facility is not running.

   -brief, -bf
   suppresses the printing of the headers. Only used in conjunction with one of the optional_args.

2. optional_args
   specifies that only selected users are to be listed and can be one of the following:

   Person_id
   lists a count of logged in users with the name Person_id.

   .Project_id
   lists a count of logged in users with the project name Project_id.

   Person_id.Project_id
   lists a count of logged in users with the name and project of Person_id.Project_id. The star convention is allowed in Person_id and Project_id.

Notes

If this command is invoked without any arguments, basic summary information is printed (see the first example below).

Absentee counts in a selective use of how_many_users (i.e., when an optional_arg is specified) are denoted by an asterisk (*).

Up to 20 classes of selected users are permitted.

Examples

To print summary information, type:

```
! how_many_users

Multics 2.0, load 5.0/50.0; 6 users
```

To print summary information on absentee users, type:

```
! how_many_users -absentee

Absentee users 0/2
```

To print the additional information provided by the -long control argument, type:

```
! how_many_users -long

Multics 2.0: PCO, Phoenix, Az.
Load = 13.0 out of 110.0 Units; users = 13
Absentee users = 0; Max absentee users = 45
System up since 06/25/74  0522.7
Last shutdown was at 06/25/74  0515.2
```

To print brief information about the SysDaemon project, type:

```
! how_many_users -bf .SysDaemon

SysDaemon = 3 + 0*
```

To print brief information about the user whose Person_id is Smith, type:

```
! how_many_users -bf Smith

Smith = 1 + 1*
```

Name: info

The info command prints the entryname of the segment of temporary text, the date, time, day of the week, and the user's Person_id and Project_id. The command also prints the amount of money spent by the user since the beginning of the billing period, the spending limit imposed on the user for the billing period, the number of records used in the working directory, and the quota limit of the working directory in records. (A record is a unit of disk allocation that contains 4096 characters.)

Usage

```
info
```

Example

```
!    new test.basic
     r 1458

!    info
     "text.basic" 02/19/76  1458.4 mst Thu BJones.Work
     $133.49 spent/200 limit  49 records used/80 limit
     r 1459
```

input

Name: input

The input command provides a convenient way to enter line numbered text. The system types the line number and one space. The user completes the line. This input mode is terminated when the user types a newline character (ASCII code 012) without having typed any other characters on the line.

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

Usage

input {new_number increment}

where:

1. new_number
   is the number of the first new line. If this number is not given, lines are entered at the end of the temporary text.

2. increment
   is the increment to use to derive subsequent line numbers (10 by default). This argument can only be specified if the new_number argument is given.

Example

```
!    new test.basic
     r 0935

!    input
     00100 !let a = 10.2
     00110 !let x = 3.9
     00120 !                    /newline
     r 0937

!    print_text

     test.basic  01/22/76 0937.6 mst Fri

     00100 let a = 10.2
     00110 let x = 3.9
     r 0938
```

link

Name: link, lk

The link command causes a storage system link with a specified name to be created in a specified directory pointing to a specified segment or directory. For a discussion of links, see Section 3.

Usage

link path1_1 {path2_1 ... path1_n path2_n}

where:

1. path1_i
   specifies the pathname of the segment to which path2_i is to point. The pathnames must be specified in pairs.

2. path2_i
   specifies the pathname of the link to be created. If the last path2_i is not specified, a link to path1_i is created in the working directory with the entryname portion of path1_i as its entryname.

Notes

The user must have append permission for the directory in which the link is to be created.

Entrynames must be unique within the directory. If the creation of a specified link would introduce a duplication of names within the directory, and if the old entry has only one name, the user is interrogated as to whether he wishes the entry bearing the old instance of the name to be deleted. If he answers "no", the link is not created. If the old entry has multiple names, the conflicting name is removed and a message to that effect is issued to the user. In either case, since the directory in which the link is to be created is being changed, the user must also have modify permission for that directory.

The star and equal conventions can be used.

Example

The command line:

```
link >my_dir>beta  alpha  >dictionary>grammar
```

creates two links in the working directory, named alpha and grammar; the first points to the segment beta in the directory >my_dir and the second points to the segment grammar in the directory >dictionary.

list

Name: list, ls

The list command prints information about entries contained in a single directory. A large selection of control arguments enables the user to specify the directory to be listed, which entries are to be listed, the amount and kind of information to be printed for each entry, and the order in which the entries are to be listed.

Usage

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

where:

1. entrynames
   are the names of entries to be listed. If entrynames are given, only entries having at least one name matching an entryname argument are listed. The star convention can be used. If no entryname argument is given, all entries (of the types specified by control arguments) in the directory are listed. A pathname can be given instead of an entryname. In this case, entries matching the entryname portion of the pathname, in the directory specified by the directory portion of the pathname, are listed. See the description of the -pathname control argument for restrictions on the use of this feature.

2. control_args
   can be chosen from the arguments described in "Control Arguments for the list Command" below.

Except where otherwise noted in the descriptions of the control arguments ("Control Arguments for the list Command" below), the entrynames and control_args arguments can appear anywhere on the command line.

Basic Use of the list Command

If the list command is invoked without any arguments, it lists all segments and multisegment files in the working directory, printing the name(s), access mode, and length of each. Segments and multisegment files are listed separately (segments first), each preceded by a line giving the total entries of that type and the sum of their lengths. (This line is referred to as the totals information or the header.) Within each entry type, entries are listed in the order in which they are found in the directory.

The following example shows the result of invoking the list command without any arguments (the line typed by the user is preceded by an exclamation mark):

```
! list

Segments = 8, Lengths = 41.

r w   10  new_code_info.runout
rew    9  new_code_info.runoff
r w    3  work.pl1
r w    7  work.list
re     2  work
r w    1  print.ec
r w    1  output_file
r      8  data_base

Multisegment-files = 1, Lengths = 334.

r w  334  info_segs
```

Directories and links are not listed by default. Notice that the information about the entries is arranged in columns without column headings. The set of columns printed by the list command depends on the control arguments given by the user and the type of entry being listed.

There are four entry types: segments, multisegment files, directories, and links. Segments and multisegment files are referred to collectively as files; segments, multisegment files, and directories are referred to collectively as branches. The set of possible columns is different for branches and links. For branches, the set of possible columns and their order (from left to right) is: modification date, date and time used, access mode, size, names, and number of names; for links: date and time entry modified, names, number of names, and link pathname. The modification-date column contains either the date and time the entry was modified or the date and time the contents were modified, and the size column contains either records used or length (in records) computed from the bit count, as specified by control arguments. Unless otherwise specified by control arguments, the items printed for branches are: access modes, length, and names; for links: names and link pathname.

The list command offers the user precise control over the command output. The various control arguments specify exactly what is to be printed. Most users will find that the following subset of list command control arguments allows them to adequately define the desired information.

-file, -f
: lists information about files. (This is the default.)

-directory, -dr
: lists information about directories.

-link, -lk
: lists information about links.

-name, -nm
: prints the names column, giving primary and any additional names of each entry.

-date_time_entry_modified, -dtem
prints the date and time the entry was last modified (e.g., by the changing of attributes such as names, ACL, or bit count).

-pri
prints only the primary name (in the names column) of each entry.

-sort XX, -sr XX
sorts the entries, within each entry type, according to the column name specified by XX. (The column names and their sorting order are described under "Entry Order" below.)

-total, -tt
prints only the heading line (totals information) for each entry type specified; this line gives the total number of entries and the sum of their sizes.

Detailed information on each of the above control arguments is given in "Control Arguments for the list Command" below.

Control Arguments for the list Command

The control arguments for the list command are described in detail on the following pages. For convenience, these arguments have been arranged in categories according to the function they perform. The categories and their respective control arguments are listed below (detailed descriptions follow the list, in the same order):

directory
-pathname path, -pn path

entry type
-segment, -sm
-multisegment_file, -msf
-file, -f
-directory, -dr
-branch, -br
-link, -lk
-all, -a

columns
-mode, -md
-length, -ln
-record, -rec
-name, -nm
-date_time_used, -dtu
-date_time_entry_modified, -dtem
-date_time_contents_modified, -dtcm
-count, -ct
-link_path, -lp

totals/header line
-total, -tt
-no_header, -nhe

multiple-name entries
-primary, -pri
-match

entry order
-sort XX, -sr XX
-reverse, -rv

entry exclusion
-exclude entryname, -ex entryname
-first N, -ft N
-from D, -fm D
-to D

output format
-brief, -bf
-short, -sh

DIRECTORY

If no directory is specified, the working directory is assumed. The list command can list only one directory at a time, and it is an error to specify more than one directory to be listed.

-pathname path, -pn path
causes entries in the directory specified by path to be listed.

The directory to be listed can also be specified by giving a pathname instead of an entryname argument, as described earlier. The difference between the two methods of specifying a directory is that the entire pathname after the -pathname control argument is taken to be that of a directory whose entries are to be listed, while a pathname not preceded by the -pathname control argument is separated into its directory and entryname portions, and the former specifies the directory while the latter specifies the entries within it that are to be listed.

ENTRY TYPE

If no control argument from this category is given, the -file control argument is assumed. More than one of the following control arguments can be given.

-segment, -sm
lists information about segments.

-multisegment_file, -msf
lists information about multisegment files.

-file, -f
lists information about files (i.e., segments and multisegment files, in that order).

-directory, -dr
lists information about directories.

-branch, -br
lists information about branches (i.e., segments, multisegment files, and directories, in that order).

-link, -lk
lists information about links.

-all, -a
lists information about all entry types in the following order: segments, multisegment files, directories, and links.

COLUMNS

If no control argument from this category is given, the access-mode, length, and names columns (in that order) are printed for branches and the names and link-path columns (in that order) are printed for links. More than one of the control arguments listed below can be given in a single invocation of the list command. When the -brief, -mode, -record, -length, or -name control arguments are given, only the names column plus those columns explicitly selected by control arguments are printed.

The user is given a choice as to what can be printed in two of the columns for branches (size and modification date). For size, the user can choose between length computed from the bit count or a count of records used. For modification date, the user can choose between the date and time the entry was modified (e.g., by the changing of attributes such as names, ACL, or bit count) or the date and time the contents of the segment or directory were modified.

If sorting by a size or modification date is specified, the above choices also apply to sorting, and the specifications of what to sort on and what to print must be consistent. For example, it is not possible to print length computed from bit count while sorting on records used.

Because of the way the information is maintained by the storage system, the records-used, date-time-contents-modified, and date-time-used values are more expensive to obtain than the other items printed by the list command. It is recommended that these values not be used for printing or sorting except when absolutely necessary. Less expensive alternatives are provided that should be suitable in most cases (e.g., length computed from bit count, and date and time the entry was modified).

The names column is printed in every invocation of the list command except when the user explicitly requests only totals information (see "Totals/Header Line" below).

-mode, -md
prints the access-mode column.

-length, -ln
prints current length computed from the bit count. This control argument is inconsistent with the -record control argument; only one of the two can be given. The -length argument, which is the less expensive of the two, is the default.

-record, -rec
prints the records used. This argument is inconsistent with the -length control argument; only one of the two can be given. The -record control argument is the more expensive of the two.

-name, -nm
prints the names column, giving the primary name and any additional names of each entry.

-date_time_used, -dtu
prints the date and time the entry was last used.

-date_time_entry_modified, -dtem
prints the date and time the entry was last modified. (e.g., by the changing of attributes such as names, ACL, or bit count). This argument is inconsistent with the -date_time_contents_modified control argument; only one of the two can be given. This argument is the less expensive of the two.

-date_time_contents_modified, -dtcm
prints the date and time the contents of the segment or directory were last modified. This argument is inconsistent with the -date_time_entry_modified control argument; only one of the two can be given. This argument is the more expensive of the two.

-count, -ct
prints the count column, which gives the total number of names for entries that have more than one name.

-link_path, -lp
prints the link-path column.

TOTALS/HEADER LINE

If no control argument from this category is given, both totals and detailed information are printed.

-total, -tt
prints only the heading line (totals information) for each entry type specified; this line gives the total number of entries and the sum of their sizes.

-no_header, -nhe
omits all heading lines.

MULTIPLE-NAME ENTRIES

The control arguments in this category are applicable only to entries that have more than one name. If no control argument from this category is given, all of the names of the specified entries are printed in the names column.

-primary, -pri
prints, in the names column, only the primary name of each entry. This control argument does not suppress the printing of any other columns; it merely suppresses the printing of secondary names.

-match
prints, in the names column, only those names that match one of the given entrynames.

ENTRY ORDER

If no control argument from this category is given, entries are printed in the order in which they are found in the directory.

-sort XX, -sr XX
sorts entries, within each entry type, according to the sort column XX where XX can be one of the following:

name, nm	sort entries by primary name, according to the standard ASCII collating sequence.
length, ln	sort entries by length computed from the bit count, largest first. This argument is inconsistent with the -record control argument.
record, rec	sort entries by records used, largest first. This argument is inconsistent with the -length control argument. If this argument is given, and the size column is being printed, the value printed in that column will be records used, rather than length.
mode, md	sort entries by access mode in the following order: null, r (or s), rw (or sm), re, rew (or sma). (This order is the result of sorting by the internal representation of the mode.)
date_time_entry_modified, dtem	sort entries by the date and time the entry was last modified, most recent first. This argument is inconsistent with the -dtcm control argument. If the -dtem control argument is given and no sort key follows the -sort control argument, then this argument is implied as the default sort key.

date_time_contents_modified, dtcm	sort entries by the date and time the contents of the entry were last modified, most recent first. This argument is inconsistent with the -dtem control argument. If the -dtcm control argument is given and no sort key follows the -sort control argument, then this argument is implied as the default sort key.
date_time_used, dtu	sort entries by the date and time used, most recent first.
count, ct	sort entries by number of names, most names first.

It is not necessary for a column to be printed in order to sort on it.

If the sort column XX is omitted, the default sorting column is determined as follows: if no date column is being printed, sort by primary name; if only one of the date columns is being printed, sort by that date; if both the modification-date and date-time-used columns are being printed, sort by the modification-date column.

Links can only be sorted by the name, modification-date, or count columns. If sorting by any other column is specified, links are printed in the order in which they are found in the directory, while branches (if also being listed) are sorted by the specified column. (See "Notes" below.)

-reverse, -rv
prints entries in the reverse of the order in which they are found in the directory. If the -sort control argument is also given, the specified sort is reversed.

ENTRY EXCLUSION

The following control arguments limit the amount of output produced by excluding entries according to either name or date or by setting an upper limit on the number of entries listed.

-exclude entryname, -ex entryname
do not list any entries that have a name that matches the specified entryname. The star convention can be used.

If the user wishes to exclude more than one entryname, he must give an -exclude control argument for each one of them. The entrynames given in all -exclude control arguments and any names given in the entryname arguments (explained on the first page of the list command description) operate together to limit the entries that are listed. All entries that have at least one name that matches any one of the entrynames given in the -exclude control arguments are excluded from the listing. From the entries that remain, those matching any of the entryname arguments are listed; if no entryname arguments are given, all the remaining entries are listed. (See "Examples" below.)

-first N, -ft N
: list only the first N entries (after sorting, if specified) of each entry type being listed. The heading lines contain the totals figures for all entries that would have been listed if the -first control argument had not been given. This argument is useful to avoid tying up a terminal by listing a large directory, when only the first few entries are of interest.

The following two arguments exclude entries on the basis of date. The date used in this comparison is the modification-date value in all cases except when the only date column being printed or sorted on is the date-time-used column. If no date column is being printed, the date-time-entry-modified value is used.

-from D, -fm D
: do not list any entries that have a date value (selected as described above) before the one specified by D.

-to D
: do not list any entries that have a date value (selected as described above) after the one specified by D.

The D value after the -from or -to control arguments must be a string acceptable to the convert_date_to_binary_ subroutine, described in the MPM Subroutines. If the date-time string contains spaces, the string must be enclosed in quotation marks. The D value can specify both a date and a time; if only a date is given, then the convert_date_to_binary_ subroutine uses, as the default time, the current time of day.

If both the -from and -to control arguments are given, the -from D value must be earlier than the -to D value.

OUTPUT FORMAT

If no control argument from this category is given, the output format of the list command is not changed.

-brief, -bf
: if just totals information is being printed, this argument causes the totals information for all selected entry types to be abbreviated and printed on a single line. Otherwise, it suppresses the printing of the default columns when they are not explicitly named in control arguments. For example, typing:

```
list -dtu -brief
```

causes the names and date-time-used columns, but not the access-mode and length columns, to be printed.

-short, -sh
: prints link pathnames starting two spaces after their entrynames, instead of aligning them in column position 35.

Notes

The obsolete name for a modification date (date_time_modified, dtm) is accepted, in both the control argument and sort key form, as a synonym for the date-time-entry-modified value.

Links do not have a date-time-contents-modified value. If links are being listed and either modification-date value is specified for printing, sorting, or entry exclusion (using the -from and -to control arguments), the date-time-entry-modified value of links is used.

Examples

The command line:

```
list -primary -count
```

lists all files in the working directory (the default directory); the names column contains only the primary names of all entries; the total number of names (for those entries having more than one name) is printed after the primary name. In addition to the names column, the access-mode and length columns are printed.

The command line:

```
list -exclude *.*
```

lists all the files in the working directory having other than two-component names, printing the three default columns (access mode, length, and names).

The command line:

```
list -segment *.* -exclude *.pl1
```

lists all the segments in the working directory having two-component names whose second component is not pl1, printing the three default columns.

The command line:

```
list -date_time_entry_modified -sort
```

lists all files in the working directory, sorted by the date-time-entry-modified column (the default sort key since the user specifically requested that date column). The date-time-entry-modified column is printed in addition to the three default columns.

The command line:

```
list -name -sort date_time_modified
```

lists all files in the working directory, sorted by the date-time-entry-modified value. Only the names column is printed. Note the use of date_time_modified as a synonym for date_time_entry_modified.

The command line:

```
list -segment -name -primary -no_header
```

lists only the primary name of each segment in the working directory without printing the heading line or any blank lines. This combination of arguments, together with the file_output command, is useful for generating a file that contains the primary names of a selected set of entries.

The command line:

```
list -mode -primary
```

lists the access mode and primary name of each file in the working directory.

The command line:

```
list -total -to "7/1/75 0000.0" -dtu -rec
```

prints the totals (number of entries and total records used) for all files that have not been used since the end of June 1975. Notice that the -dtu control argument is used to specify that the -to date refers to the date and time used.

list_acl

Name: list_acl, la

The list_acl command lists the access control lists (ACLs) of segments, multisegment files, and directories. For a description of ACLs, see Section 4.

Usage

list_acl {path} {User_ids} {-control_args}

where:

1. path
   is the pathname of a segment, multisegment file, or directory. If it is -wd, -working_dir, or omitted, the working directory is assumed. If it is omitted, no User_ids can be specified. The star convention can be used.

2. User_ids
   are access control names that must be of the form Person_id.Project_id.tag. All ACL entries with matching names are listed. (For a description of the matching strategy, refer to the set_acl command.) If User_id is omitted, the entire ACL is listed.

3. control_args
   can be chosen from the following control arguments:

   -ring_brackets, -rb
   lists the ring brackets. This control argument can appear anywhere on the line and affects the whole line. Ring brackets are discussed under "Intraprocess Access Control" in Section 6 of the MPM Reference Guide.

   -brief, -bf
   suppresses the message "User name not on ACL of path."

   -directory, -dr
   lists the ACLs of directories only. The default is segments, multisegment files, and directories. (See "Notes" below.)

   -segment, -sm
   lists the ACLs of segments and multisegment files only.

Notes

The -directory and -segment control arguments are used to resolve an ambiguous choice that may occur when path is a star name.

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

Active Function Usage

[list_acl {path} {User_id} {-control_args}]

where the arguments are the same as above.

Example

The command line:

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

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

The command line:

```
list_acl *.pl1  -rb
```

lists the whole ACL and the ring brackets of every segment in the working directory that has a two-component name with a second component of pl1.

The command line:

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

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

Name: locate, l

The locate command searches the temporary text for all occurrences of a specified character string. Each line containing the string is printed. The entire line, including the line number, is used in matching the string.

Usage

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

where:

1. /
   is any delimiter not found in string, except blank, tab, or a digit.

2. string
   is a string of characters to be found.

3. first_line
   is the line number of the first line to be searched. If this argument is missing, the entire text is searched.

4. last_line
   is the line number of the last line to be searched; if this argument is not given, the search is made from the first line specified by the first_line argument to the end of the text. This argument can only be given if the first_line argument is given.

Note

All lines between and including the first and last lines specified are located. The line numbers specified by the first_line and last_line arguments do not have to appear in the text, but the range specified by them must contain at least one line.

Example

```
!    130 for n = 1 to 5
!    140 let e = 40
!    150 for m = 1 to 3
!    160 let e = e + p(m)
!    locate /m/                  /entire text is searched
     150 for m = 1 to 3
     160 let e = e + p(m)
     r 1246


!    locate /e/ 135 160
     140 let e = 40
     160 let e = e + p(m)
     r 1247
```

login

Name: login, l

The login request is used to gain access to the system. It is a request to the answering service to start the user identification and process creation procedures.

The login request asks for 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 digits associated with the Person_id.

After the user responds with the password, the answering service looks up the Person_id, the Project_id, and the password in its tables and verifies that the Person_id is valid, that the Project_id is valid, that the user is a legal user of the project, and that the password given matches the registered password. If these tests succeed, the load control mechanism is consulted to determine if allowing the user to log in would overload the system.

If the user is permitted to log in, a process is created for the user, and the terminal is placed under control of that process.

Many control arguments are available for tailoring various attributes of the new process to the user's needs.

Usage

login Person_id {Project_id} {-control_args}

where:

1. Person_id
   is the user's registered personal identifier. This argument must be supplied. The user's personal identifier can be replaced by his registered "login alias" if he has one. Aliases, like personal identifiers, are registered by the system administration and are unique at the installation. The login alias is translated into the user's personal identifier during the login process, and there is no difference between a user process created by supplying a personal identifier and one created by supplying an alias.

2. Project_id
   is the identification of the user's project. If this argument is not supplied, the default project associated with the Person_id is used. See the -change_default_project control argument below for changing the default project to the Project_id specified by this argument.

3. control_args
can be selected from the following:

-authorization STR, -auth STR
set the authorization of the process to that specified by STR, where STR is a character string composed of level and category names for the desired authorization, separated by commas. The STR character string may not contain any embedded blank or tab characters. (The short names for each level and category are guaranteed to not contain any blanks or tabs, and can be used whenever the corresponding long names do contain blanks or tabs.) The STR character string must represent an authorization that is less than or equal to the maximum authorization of Person_id on the project Project_id. If this control argument is omitted, the user's registered default login authorization is used. (See "Authorizations" in Section 3 for more information about process authorizations.)

-brief, -bf
suppress messages associated with a successful login. If the standard process overseer is being used, then the message of the day is not printed.

-change_default_auth, -cda
change the user's registered default login authorization to the authorization specified by the -authorization control argument. If the authorization given by the user is valid, the default authorization is changed for subsequent logins, and the message "default authorization changed" is printed at the terminal. If the -cda control argument is given without the -auth argument, an error message is printed.

-change_default_project, -cdp
change the user's default project to be the Project_id specified in this login request line (see the description of the Project_id argument above). The default Project_id is changed for subsequent logins, and the message "default project changed" is printed at the user's terminal. If the -cdp control argument is given without a Project_id argument, an error message is printed.

-change_password, -cpw
change the user's password to a newly given password. The login request asks for the old password before it requests the new one. It requests the new one twice, to verify the spelling. If it is not typed the same both times, the login and the password change are refused. If the old password is correct, the new password replaces the old for subsequent logins, and the message "password changed" is printed at the user's terminal. The user should not type the new password as part of the control argument.

-force
log the user in if at all possible, provided the user has the guaranteed login attribute. Only system users who perform emergency repair functions have the necessary attribute.

-generate_password, -gpw
change the user's password to a new password, generated for the user by the system. The login request asks for the old password first. Then, a new password is generated and typed on the user's terminal. The user is asked to retype the new password, to verify that he has seen it. If the user types the new password correctly, it replaces the old password for subsequent logins, and the message "password changed" is printed at the user's terminal. If the user mistypes the new password, the login and password change are refused.

-home_dir path, -hd path
set the user's home directory to the path specified, if the user's project administrator allows him to specify his home directory.

-modes STR, -mode STR, -md STR
set the I/O modes associated with the user's terminal to STR, where the string STR consists of modes acceptable to the tty_ I/O module. (See the tty_ I/O module description in the MPM Subroutines, Order No. AK93-02B, for a complete explanation of possible modes.) The STR string is usually a list of modes separated by commas; the STR string must not contain blanks. (See "Examples" below.)

-no_preempt, -np
refuse to log the user in if he can only be logged in by preempting some other user in his load control group.

-no_print_off, -npf
cause the system to overtype a string of characters to provide a black area for the user to type his password.

-no_start_up, -ns
instruct the standard process overseer not to execute the user's start_up.ec segment, if he has one, and if the project administrator allows him to avoid it.

-no_warning, -nw
suppress even urgent system warning and emergency messages from the operator, both at login and during the user's session. Use of this argument is recommended only for users who are using a remote computer to simulate a terminal, or are typing out long memoranda, when the process output should not be interrupted by even the most serious messages.

-outer_module p, -om p
attach the user's terminal via the outer module named p rather than the user's registered outer module, if the user has the privilege of specifying his outer module.

-print_off, -pf
suppress overtyping for the password.

-process_overseer path, -po path
set the user's process overseer to the procedure given by the path specified, if the user's project administrator allows him to specify his process overseer. If path ends in the characters ",direct", the specified procedure is called directly during process initialization rather than by the standard procedure provided by the system. This means that the program specified by path must perform the tasks that would have been performed by the standard procedure.

login

-ring N, -rg N
set the user's initial ring to be ring N, if this ring number is greater than or equal to the user's registered initial ring and less than his registered maximum ring.

-subsystem path, -ss path
create the user's process using the prelinked subsystem in the directory specified by path. The permission to specify a process overseer, which can be given by the user's project administrator, also governs the use of the -subsystem argument. To override a default subsystem specified by the project administrator, type -ss "".

-terminal_type STR, -ttp STR
set the user's terminal type to STR, where STR is any terminal type name defined in the standard terminal type table. (To obtain a list of terminal types, refer to the print_terminal_types command.) This control argument overrides the default terminal type.

Notes

If neither the -print_off nor -no_print_off control argument is specified at log-in, the system attempts to choose the option most appropriate for the user terminal type.

Several parameters of the user's process, as noted above, can be controlled by the user's project administrator. The project administrator can allow the user to override some of these attributes by specifying control arguments in his login line.

If the project administrator does not allow the user to specify the -subsystem, -outer_module, -home_dir, -process_overseer, or -ring control arguments or if he does allow one or more of these control arguments and they are incorrectly specified by the user, a message is printed and the login is refused.

Examples

In the examples below, the lines typed by the user are preceded by an exclamation mark (!) and the user's password is shown even though in most cases the system 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 Project_id (and then the password) as:

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

To set (or change) the default project to Demo, type:

```
! login Jones Demo -cdp
  Password:
! mypass
  Default project changed.
```

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 (assuming the user has a default project), type:

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

To change the password from mypass to newpass (assuming the user has a default project), type:

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

logout

Name: logout

The logout command terminates a user session and ends communication with the Multics system. It signals the finish condition for the process; and, after the default on unit for the finish condition closes all open files and returns, it destroys the process.

Usage

logout {-control_args}

where control_args can be chosen from the following:

-hold, -hd
the user's session is terminated. However, communication with the Multics system is not terminated, and a user can immediately log in without redialing.

-brief, -bf
no logout message is printed, and if the -hold control argument has been specified, no login message is printed either.

Note

See Section 2, "How to Access the Multics System" in the Multics Users' Guide, Order No. AL40.

merge_text

Name: merge_text, mgt

The merge_text command inserts the contents of a segment into the temporary text. To avoid line number duplication, the resulting segment is resequenced starting at the first line of merged text.

Special editing is done for BASIC source text. Any references to the lines that are renumbered are edited to reflect the new numbers. This editing is done only if the entryname of the segment of temporary text ends with the basic suffix.

Usage

```
merge_text  path {line_number}
```

where:

1. path
   is the pathname of the segment to be inserted. This segment must contain line numbered text.

2. line_number
   specifies the line after which the segment is inserted. If no line_number argument is given, the segment is inserted at the end of the temporary text.

Example

```
!    print_text check          /segment to be inserted

     check  01/17/76  111.2 mst Fri

     100 if x > y then 150
     120 let w = y
     130 let y = x
     140 let x = w
     150 call "trans": x,y
     r 1112


!    print_text -nhe           /prints temporary text
     10 input x,y,z
     12 goto 10
     r 1113


!    merge check 10
     r 1113
```

```
!     print_text -nhe
      10 input x,y,z
      20 if x > y then 60      /special editing by merge_text of reference to
                               /reflect new line numbers
      30 let w = y
      40 let y = x
      50 let x = w
      60 call "trans": x,y
      70 goto 10               /this number was changed to prevent
      r 1114                   /overlap
```

Name: move_text, mt

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

Special editing is done for BASIC source text. Any references to the lines that are renumbered are edited to reflect the new numbers. This editing is done only if the entryname of the segment of temporary text ends with the basic suffix.

Usage

```
move_text firstline {last_line},line_number
```

where:

1. first_line
   is the line number of the first line to be moved.

2. last_line
   is the line number of the last line to be moved. If this argument is not given, only the line specified by the first_line argument is moved.

3. line_number
   specifies the line number after which the moved lines are to be inserted. The line number specified must be preceded by a comma.

Note

All lines between and including the first and last lines specified are relocated. The line numbers specified by the first_line and last_line arguments do not have to appear in the text, but the range specified by them must contain at least one line.

Example

```
!     print_text

      tmst.basic  01/17/76  1313 mst Fri

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

      r 1314


!     move_text 140 155,600      /moves all lines between and including
      r 1315                     /lines 140 and 155


!     print_text -nhe
      100 if x > m then 160
      110 if x < 0 then 610     /140 changed to 610 by move_text
      120 let t = t + x
      130 goto 100
      160 gosub 300
      610 print "illegal x"     /location following line 600
      620 stop
      r 1316
```

Name: new

The new command deletes the temporary text and creates a pathname for the new segment of temporary text.

Usage

```
new {path}
```

where path is the pathname of the newly created segment that will contain the temporary text. If path is omitted, the name of the segment of temporary text is set to null (""). (See "Segment Naming Conventions" in Section 1 for a description of valid segment names.)

Note

The path argument given with the new command is used by default if the save, print_text, and info commands are given without a path argument.

Examples

```
!    new
!    10 Jan
!    20 Susan
!    30 Betsy
!    40 Sarah
!    50 Lilli
!    print_text

     "" 03/20/76 1016.2 mst Sat

     10 Jan
     20 Susan
     30 Betsy
     40 Sarah
     50 Lilli
     r 1016
```

new

```
!     new multiply.fortran
!     10 input 100, x,y
!     20 z = x*y
!     30 print 100,z
!     32 100 format (f5.0,f5.0)
!     40 end
!     print_text

      multiply.fortran  03/20/76  1017.6  mst Sat

      10 input 100, x,y
      20 let z = x*y
      30 print 100,z
      32 100 format (f5.0,f5.0)
      40 end
      r 1017
```

Name: old

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

Usage

old path

where path is the pathname of the segment to be retrieved.

Note

Segments retrieved from other directories can be saved in the user's working directory by issuing the save command with a segment name as an argument to the command.

Examples

```
!    old eval.fortran          /the segment named eval.fortran in the working
     r 1534                    /directory becomes the temporary text
!    save eval2.fortran        /saves the temporary text in the working directory
     r 1534                    /with the name eval2.fortran


!    old >udd>Design>Smith>summary.basic   /the segment named summary.basic in
     r 1535                                /the Smith directory becomes the
                                           /temporary text
!    save summary.basic        /the temporary text is saved in the working
     r 1535                    /directory with the name summary.basic


!    old >udd>Design>Jones>junk
     r 1536
!    save                      /the temporary text is saved in the Jones
     r 1536                    /directory with the name given with the old command


!    old >udd>student_lib>sort.fortran
     old:  segment not found  ">udd>student_lib>sort.fortran"
     r 1537
```

Name: print_text, pt

The print_text command prints one or more lines of a segment.

Usage

print_text {path} {-control_arg} {first_line last_line}

where:

1. path
   is the pathname of the segment to be printed. If path is not given, the temporary text is assumed.

2. control_args
   is one or more of the following:

   -pathname path, -pn path
   specifies the segment to print. This argument is only needed if the segment name begins with a digit.

   -no_header, -nhe
   suppresses all heading lines.

3. first_line
   is the line number of the line at which to begin printing.

4. last_line
   is the line number of the line at which to stop printing. This argument can only be given if the first_line argument is given.

Notes

If print_text is invoked without any arguments, all of the temporary text is printed, preceded by heading lines (one line giving the name of the segment and current date information and one blank line). These heading lines are suppressed whenever the first_line argument is supplied.

If only the first_line argument is specified, only that line is printed.

All lines between and including the first and last lines specified are printed. The line numbers specified by the first_line and last_line arguments do not have to appear in the text but the range specified by them must contain at least one line.

print_text

Examples

```
!     print_text

      min.basic  11/07/76  1211  mst Fri

      105 if x < y  then 140
      120 if x < z then 150
      130 let z = x + y
      140 print x, y, z
      r 1212


!     print_text 105
      105 if x < y then 140
      r 1213


!     print_text 120 140
      120 if x < z then 150
      130 let z = x + y
      140 print x, y, z
      r 1216
```

Name: ready_off, rdf

The ready_off command suppresses the system ready message. See the ready_on command.

Usage

```
ready_off
```

Examples

```
!    print_text 10 20
     10 print "totals", "average"
     12 input x
     14 if x < 0 then 35
     r 1833

!    ready_off
!    print_text 10 20
     10 print "totals", "average"
     12 input x
     14 if x < 0 then 35
!    save
1    ready_on
     r 1834
```

Name: ready_on, rdn

The ready_on command restores the system ready message. Since ready messages are printed by default, the ready_on command is only needed to "cancel" a previously issued ready_off command.

Usage

```
ready_on
```

Example

```
1     ready_off
!     print_text -nhe
      10 input x,y
      20 let z = x+y
      30 print z
      40 end
!     ready_on
      r 1758
```

Name: rename, rn

The rename command replaces a specified segment, multisegment file, directory, or link name by a specified new name, without affecting any other names the entry might have.

Usage

rename {-control_arg} path1 name1 {... {-control_arg} pathn namen}

where:

1. pathi
   specifies the old name that is to be replaced; it can be a pathname or an entryname.

2. namei
   specifies the new name that replaces the storage system entryname portion of pathi.

3. control_arg
   can be -name or -nm to indicate that the path argument that follows it is an entryname containing special command system symbols (e.g., < or *) that are not interpreted in the usual way by the command processor. Thus, this control argument allows the user to rename strangely named segments.

Notes

The star and equal conventions can be used; however, these conventions are suppressed when the -name control argument is used.

The access mode of the user with respect to the directory specified by pathi must contain the modify attribute.

Since two entries in a directory cannot have the same entryname, special action is taken by this command if namei already exists in the directory specified by pathi. If the entry having the entryname namei has an alternate name, entryname namei is removed and the user is informed of this action; the renaming operation then takes place. If the entry having the entryname namei has only one name, the entry must be deleted in order to remove the name. The user is asked if the deletion should be done; if the user answers "no", the renaming operation does not take place.

Example

The command line:

```
rename alpha beta >sample_dir>gamma delta
```

renames alpha, in the user's working directory, to beta and also renames gamma, in the directory >sample_dir, to delta.

The command line:

```
rename -name *stuff junk
```

renames the segment *stuff, in the working directory, to junk.

Name: resequence, rsq

The resequence command renumbers specified lines of temporary text, beginning with a given line number and adding a given increment to derive subsequent numbers.

This command does special editing for BASIC source text. Any reference to the lines that are renumbered are edited to reflect the new numbers. In all other cases, only the line numbers at the beginning of the line are changed. This editing is done only if the entryname of the segment of temporary text ends with the basic suffix.

Usage

resequence {new_number increment}

where:

1. new_number
   is the first new line number to be assigned (100 by default).

2. increment
   is the increment used to derive subsequent line numbers (10 by default). This argument can only be specified if the new_number argument is given.

Example

```
!    old prog.basic
     r 1617


!    print_text -nhe
!    210 if m>n then 260
!    220 next i
!    230 if n<>m then 260
!    240 print "ok"
!    250 stop
!    260 go to 400
!    resequence
     r 1618

!    print_text -nhe
     100 if m>n then 150
     110 next i
     120 if n<>m then 150
     130 print "ok"
     140 stop
     150 go to 400
     r 1619
```

run

Name: run

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

Usage

run {path}

where path is the pathname of a segment. If path is not given, the run command compiles the temporary text and executes it provided that the entryname of the temporary text has a language suffix. If path is specified and the entryname has a language suffix, the run command expects the segment to contain the source program and compiles and executes it. If path is specified and the entryname does not contain a language suffix, the run command expects the segment to contain object code and exeuctes it.

Examples

```
!    old test.basic
     r 1721

!    run
     (program execution of text.basic)
     r 1721

!    run std          /object segment "std" in working directory
     (program execution)
     r 1722.3

!    run              /temporary text is not changed by run
     (program execution of test.basic)
     r 1723
```

Pages 96 through 98 of this section have been deleted.

Name: save

The save command causes the temporary text to be saved in the user's working directory.

Usage

save {path}

where path is the pathname under which the temporary text is to be saved. If no argument is supplied, the temporary text is saved using the pathname given with the last new, old, or save command (see "Note" below). If there has been no preceding new, old, or save command, an error message is printed.

Note

The save command overwrites a previously saved segment of the same name.

Examples

```
!    save newprog.fortran
     r 1417

!    old scores.basic
     r 1418

!    10 data 87,93,78,40
!    save          /the name from the old command is used
     r 1419
```

Name: set_acl, sa

The set_acl command manipulates the access control lists (ACLs) of segments, multisegment files, and directories. See Section 4 for a discussion of ACLs.

Usage

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

where:

1. path
   is the pathname of a segment, multisegment file, or directory. If it is -wd or -working_dir, the working directory is assumed. The star convention can be used and applies to either segments and multisegment files or directories, depending on the type of mode specified in mode1.

2. modei
   is a valid access mode. For segments or multisegment files, any or all of the letters rew; for directories, any or all of the letters sma with the requirement that if modify is present, status must also be present. 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 Person_id of the user and current Project_id are assumed.

4. control_args
   can be either of the following control arguments:

   -directory, -dr
   specifies that only directories are affected.

   -segment, -sm
   specifies that only segments and multisegment files are affected. This is the default.

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

Notes

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

The user needs modify permission on the containing directory.

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

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

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

..* matches only the literal ACL entry "*.*.*".

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

JRSmith.. matches any ACL entry with a first component of JRSmith.

.. matches any ACL entry.

. matches any ACL entry with a last component of *.

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

Examples

The command line:

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

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

The command line:

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

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

set_acl

The command line:

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 read.

set_tty

Name: set_tty, stty

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

Usage

set_tty {-control_args}

where control_args can 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. 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≤n≤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.
lfecho, ^lfecho	lfecho specifies that a linefeed is to be echoed when a carriage return is typed.
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 set in the default modes string for the current terminal type.

-tabs
specifies that the device has software-settable tabs, and that the tabs are to be set. This control argument currently has effect only for GE TermiNet 300-like devices.

-print
causes the terminal type and modes to be printed on the terminal. If any other control arguments are specified, the type and modes printed reflect the result of the command.

Notes

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

1. If the -terminal_type control argument is specified, set the specified device type and turn on the default modes for that type.
2. If the -reset control argument is specified, turn off all modes that are not set in the default modes string for the current terminal type.
3. If the -modes control argument is specified, turn on or off those modes explicitly specified.
4. If the -tabs control argument is specified, and the terminal has settable tabs, set the tabs.
5. If the -print control argument is specified, print the type and modes on the terminal.

Examples

In the following example, a user of a TermniNet 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 wll begin with "".

```
!    set_tty -modes 1160
```

Name: truncate, tc

The truncate command truncates a segment to a specified length and resets the bit count accordingly, setting the bit count author to be the user who invoked the command. The segment can be specified by pathname or segment number.

Usage

truncate {-control_arg} seg_no length

where:

1. control_arg
   if present, must be -name or -nm, indicating that the following seg_no is in fact a pathname, although it might look like a number.

2. seg_no
   is either a pathname or an octal segment number. A pathname that happens to be an octal number should be preceded by the control argument -name or -nm.

3. length
   is an octal integer indicating the length of the segment in words after truncation. If no length argument is provided, zero is assumed.

Notes

The user must have write access on the segment to be truncated.

If the segment is already shorter than the specified length, its length is unchanged, but the bit count is set to the specified length.

This command should not be used on segments that are (or are components of) structured files.

Example

The command line:

truncate alpha 50

truncates segment alpha to 50 words; i.e., all words from word 50 (octal) on are zero. The bit count of the segment is set to the truncated length.

Name: unlink, ul

The unlink command deletes the specified link entry. For a discussion of links see Section 3.

Usage

unlink paths {-control_args}

where:

1. paths
   specify storage system link entries to be deleted. The star convention can be used.

2. control_args
   can be chosen from the following:

   -force
   suppresses the query "Do you want to unlink ** in <dir_path>?" when appropriate.

   -brief, -bf
   inhibits the printing of an error message if the segment or multisegment to be deleted is not found.

Notes

The user must have modify permission on the directory containing the link.

The delete, delete_force, and delete_dir commands can be used to delete segment and directory entries.

SECTION 6

MULTICS edm TEXT EDITOR

A simple Multics context editor, edm, is used for creating and editing ASCII segments. To invoke edm, the user types:

```
edm pathname
```

where pathname identifies the segment to be either edited or created.

The edm editor operates in one of two principal modes: edit or input. If pathname identifies a segment that is already in existence, edm begins in edit mode. If pathname identifies a segment that does not exist, or if pathname is not given, edm begins in input mode. The user can change from one mode to the other by issuing the mode change character: a period (followed by a "carriage return") when this is the only character on a line. For verification, edm announces its mode by responding "Edit." or "Input." when the mode is entered.

The edm requests assume that the segment consists of a series of lines and has a conceptual pointer to indicate the current line. (The "top" and "bottom" lines of the segment are also meaningful.) Some requests explicitly or implicitly cause the pointer to be moved; other requests manipulate the line currently pointed to. Most requests are indicated by a single character, generally the first letter of the name of the request; for these requests only the single character is accepted by edm to initiate the corresponding action.

REQUESTS

Various edm requests and their indicators are listed below. Detailed descriptions of these requests are given later in this section.

-	backup
=	print current line number
,	comment mode
.	mode change
b	bottom
d	delete
f	find
i	insert
k	kill
l	locate

n	next
p	print
q	quit
r	retype
s	substitute
t	top
v	verbose
w	write

GUIDELINES

The following list offers helpful suggestions about the use of edm for the new user.

1. It is useful to remember that the editor makes all changes on a copy of the segment, not on the original. Only when the user issues a w (write) request does the editor overwrite the original segment with the edited version. If the user types q (quit) without a preceding w (write), the editor warns him that editing will be lost and the original segment will be unchanged, and gives him the option of aborting the request.

2. The user should not issue a quit signal (press ATTN, BRK, INTERRUPT, etc.) while in the editor unless he is prepared to lose all of the work he has done since the last w (write) request. However, if a quit signal is issued, the user may return to edm request level without losing his work by issuing the program_interrupt command.

3. If the user has a lot of typing or editing to do, it is wisest to occasionally issue the w request to ensure that all the work up to that time is permanently recorded. Then, if some problem should occur (with the system, the telephone line, or the terminal), the user loses only the work done since the last w request.

4. The user should be sure that he has switched from input mode to edit mode before typing editing requests, including the w and q requests. If he forgets, the editing requests are stored in the segment, instead of being acted upon. The user then has to locate and delete them.

5. As the user becomes more familiar with the use of edm, he may conclude that it provides verification responses more often than necessary, thus slowing him down. He may use the k request to "kill" the verification response. However, once the user feels confident enough to use the k request, he is probably ready to begin using the more sophisticated editor, qedx. The qedx editor provides the user with a repertoire of more concise and powerful requests, permitting more rapid work.

REQUEST DESCRIPTIONS

The following edm requests are the ones that the new user will find most useful as he begins working on Multics. Examples are included to help the new user see the practical use of the requests.

Backup (-) Request

The backup request moves the pointer backward (toward the top of the segment) the number of lines specified by the user and prints the line to show the location of the pointer. For example, if the pointer is currently at the bottom line of the following:

```
get list (n1, n2);
sum = n1 + n2;
put skip;
put list ("The sum is:", sum);
```

and the user wants the pointer at the line beginning with the word "sum," he types:

```
! -2
  sum = n1 + n2;
```

If the user does not specify a number of lines with the backup request, the pointer is moved up one line. (Typing a space between the backup request and the integer is optional.)

Print Current Line Number (=) Request

The print current line number request tells the user the number of the line the pointer is currently pointing to (all the lines in a segment are implicitly numbered by the system--1, 2, 3,..., n).

Whenever the user wants to check the implicit line number of the current line, he issues this request and edm responds with a line number.

```
! =
143
```

Comment Mode (,) Request

When the user invokes the comment mode request, edm starts printing at the current line and continues printing all the lines in the segment in comment mode until it reaches the end of the segment or until the user types the mode change character (a period) as the only entry on a line.

To print the lines in comment mode means that edm prints the line without the carriage return, switches to input mode, and waits for the user's comment entry for that line. When the user gives his comment line and a carriage return, edm repeats the process with the next line.

If the user has no comment for a particular line, he types only a carriage return and edm prints the next line in comment mode. When the user wants to leave comment mode and return to edit mode, he types--as his comment--the mode change character (a period).

Programmers will find that the comment mode request gives them a fast and easy way to put comments in their programs.

Mode Change (.) Request

The mode change request allows the user to go from input mode to edit mode or vice versa simply by typing a period as the only character on a line. This request is also the means by which the user leaves the comment mode request and returns to edit mode.

For example, when a user finishes typing information into a segment, he must leave input mode and go to edit mode in order to issue the write (w) request and save the information.

```
! last line of segment
! .
  Edit.
! w
```

Bottom (b) Request

The bottom request moves the pointer to the end of the segment (actually sets the pointer after the last line in the segment) and switches to input mode. This request is particularly helpful when the user has a lot of information to type in input mode; if he sees some mistakes in data previously typed, he can switch to edit mode, correct the error, then issue the bottom request and continue typing his information.

```
! red
! oramge
! yellow
! green
! .
  Edit.
! -2
  oramge
! s/m/n/
  orange
! b
  Input.
! blue
```

Delete (d) Request

This request deletes the number of lines specified by the user. Deletion begins at the current line and continues according to the user's request. For example, to delete the current line plus the next five lines, the user types:

```
d6
```

If the user issues the delete request without specifying a number, only the current line is deleted. (That is, the user may type either d or d1 to delete the current line.)

After a deletion, the pointer is set to an imaginary line following the last deleted line but preceding the next nondeleted line. Thus, a change to input mode would take effect before the next nondeleted line.

Find (f) Request

The find request searches the segment for a line beginning with the character string designated by the user. The search begins at the line following the current line and continues, wrapping around the segment from bottom to top, until the string is found or until the pointer returns to the current line; however, the current line itself is not searched. If the string is not found, edm responds with the following error message:

```
edm:  Search failed.
```

If the string is found and the user is in verbose mode, edm responds by printing the first line it finds that begins with the specified string.

```
!  f If
   If the string is found and the user
```

When the user types the string, he must be careful with the spacing. A single space following the find request is not significant; however, further leading and embedded spaces are considered part of the specified string and are used in the search.

In the find request, the pointer is either set to the line found in the search or remains at the current line if the search fails. Also, if the user issues the find request without specifying a character string, edm searches for the string requested by the last find or locate (l) request.

Insert (i) Request

The insert request allows the user to place a new line of information after the current line.

If the user invokes the insert request without specifying any new text, a blank line is inserted after the current line. If the user types text after the insert request, he must be careful with the spacing. One space following the insert request is not significant, but all other leading and embedded spaces become part of the text of the new line.

For example, if the pointer is at the top line of the following:

```
sum = n1 + n2;
put list ("The sum is:", sum);
```

and the user issued the following insert request:

```
i put skip;
```

the result would be:

```
sum = n1 + n2;
put skip;
put list ("The sum is:",sum);
```

If the user wants to insert a new line at the beginning of the segment, he first issues a top (t) request and then an insert request.

Kill (k) Request

The kill request suppresses the edm responses following the change (c), find (f), locate (l), next (n), or substitute (s) requests. To restore responses to these requests, the user issues the verbose (v) request.

It is recommended that the new user not use the kill request until he is thoroughly familiar with edm. The responses given in verbose mode are helpful; they offer an immediate check for the user by allowing him to see the results of his request.

Locate (l) Request

The locate request searches the segment for a line containing a user-specified string. The locate and find (f) requests are used in a similar manner and follow the same conventions. (Refer to the find request description for details.) With the find request, edm searches for a line beginning with a specified string; with the locate request, edm searches for a line containing--anywhere--the specified string.

Next (n) Request

The next request moves the pointer toward the bottom of the segment the number of lines specified by the user. If the user invokes the next request without specifying a number, the pointer is moved down one line. When the user does specify the number of lines he wants the pointer to move, the pointer is set to the specified line. For example, if the user types:

```
n4
```

the pointer is set to the fourth line after the current line. The edm editor responds, when in verbose mode, by typing the user-specified line.

Print (p) Request

The print request prints the number of lines specified by the user, beginning with the current line, and sets the pointer to the last printed line. If the user does not specify a number of lines, only the current line is printed.

If the user wants to see the current line and the next three lines, he types:

```
! p4
  current line
  first line after current line
  second
  third
```

In edm, every segment has two imaginary null lines, one before the first text line and one after the last text line. When the user prints the entire segment, these lines are identified as "No line" and "EOF" respectively.

Quit (q) Request

The quit request is invoked by the user when he wants to exit from edm and return to command level.

For the user's convenience and protection, edm prints a warning message if the user does not issue a write (w) request to save his latest editing changes before he issues the quit request. The message reminds the user that his changes will be lost and asks if he still wishes to quit.

```
! q
edm: Changes to text since last "w" request will be lost if you quit;
do you wish to quit?
```

If the user answers by typing no, he is still in edit mode and can then issue a write request to save his work. If he instead answers by typing yes, he exits from edm and returns to command level.

Retype (r) Request

The retype request replaces the current line with a different line typed by the user.

One space between the retype request and the beginning of the new line is not significant; any other leading and embedded spaces become part of the new line. To replace the current line with a blank line, the user types the retype request and a carriage return.

Substitute (s) Request

The substitute request allows the user to change every occurrence of a particular character string with a new character string in the number of lines he indicates. If the user is in verbose mode (in which edm prints responses to certain requests), edm responds by printing each changed line. If the original character string is not found in the lines the user asked edm to search, edm responds:

```
edm:  Substitution failed.
```

For example, if the pointer is at the top line of the following:

```
get list (n1, n2);
sum = n1 + n2;
put skip;
put list ("The sum is:", sum);
```

and the user wants to search the next three lines and change the word "sum" to "total," he types:

```
! s4/sum/total/
  total = n1 + n2;
  put list ("The total is:", total);
```

The four lines searched by the editor are the current line plus the next three. (The search always begins at the current line.) If the user does not specify the number of lines he wants searched, edm only searches the current line. If the user does not specify an original string, the new string is inserted at the beginning of the specified line(s).

Notice in the example that a slash (/) was used to delimit the strings. The user may designate as the delimiter any character that does not appear in either the original or the new string.

Top (t) Request

The top request moves the pointer to an imaginary null line immediately above the first text line in the segment. (See the print request description concerning imaginary null lines in edm.)

An insert (i) request immediately following a top request allows the user to put a new text line above the "original" first text line of the segment.

Verbose (v) Request

The verbose request causes edm to print responses to the change (c), find (f), locate (l), next (n), or substitute (s) requests.

Actually, the user does not need to issue the verbose request to cause edm to print the responses; when he invokes edm, the verbose request is in effect. The only time the user needs to issue the verbose request is to cancel a previously issued kill (k) request.

Write (w) Request

The write request saves the most recent copy of a segment in a pathname specified by the user. (The pathname can be either absolute or relative.)

If the user does not specify a pathname, the segment is saved under the name used in the invocation of edm. When saving an edited segment without specifying a pathname, the original segment is overwritten (the previous contents are discarded) and the edited segment is saved under the original name.

If the user does not specify a pathname and he did not use a pathname when he invoked edm, an error message is printed and edm waits for another request. If this happens, the user should reissue the write request, specifying a pathname.

ADDITIONAL REQUESTS

In addition to those edm requests described above, some more extensive requests are available to the user as familiarity with the editor increases and more ability to manipulate text is needed. Such requests are listed below, followed by a brief description of each request.

E	execute a command line
merge	insert a segment
move	move lines of text
qt	force exit from the editor
updelete	delete all previous text lines
upwrite	save all previous text lines

Execute (E) Request

The execute request is used to issue Multics (or FAST subsystem) commands to the Multics command processor for execution. To invoke the execute request the user types:

```
! E command_line
```

Merge (merge) Request

The merge request permits the user to insert text that has previously been created into the segment presently being edited. In effect, the two segments are merged together to create one segment that the user can then continue to edit.

To invoke the merge request, the user locates the line of text after which the previously created segment is to be inserted and types:

```
! merge path
```

where path is the pathname (either relative or absolute) of the segment that is to be merged into the segment being worked on. The segment named path is then inserted after the current line and the pointer is set to "no line" following the last line of the inserted segment.

If path is not specified by the user, the pathname given in the invocation of edm is used. If a pathname is given neither in this request nor in the invocation of edm, an error message is printed and edm looks for another request.

Move (move) Request

The move request is used to move lines of text from one location in the segment being edited to another location in the same segment.

In order to use the move request, the user must determine:

1. The line number of the first line of text to be moved (with a find or locate request, then a current line number request)

2. The number of lines of text to be moved.

3. The line number after which the text is to be inserted.

The format of the move request, as typed by the user is:

move m n

where m is the number of the line at which the move of text begins, and n is the number of lines to move (including line m). If no number is specified for n, only the single line, m, is moved.

If lines of text are to be inserted at line 491 then the user issues the move request in the following form:

move 146 8

which specifies that eight lines of text, beginning at line 146, are to be inserted after the current line (determined in step 3 above) and the lines of text moved are to be deleted from their original location.

The pointer is set to "no line" following the lines moved. An insert (i) request or a change to input mode could be issued by the user to take effect immediately following the moved text.

Quitforce (qt) Request

The quitforce request to the edm text editor functions much the same as the quit request in that it may be used to exit from edm and return to command level.

If the quitforce request is invoked by the user, however, the exit from edm is immediate and no warning or query is issued to the user. Thus, if the user has forgotten to issue a write request after entering text or performing other text manipulations, these operations will not be effected. Care should be exercised, therefore, in using the quitforce request, as the user may find that this shortcut has cost the loss of previously expended time.

Delete to Pointer (updelete) Request

The updelete request is used to delete all of the lines that precede (but not including) the current line. Thus, if the user decided that the first 100 lines of the segment being worked on were not needed any longer, he would move to line 101 and issue the request:

! updelete

to delete the first 100 lines of the segment.

Write to Pointer (upwrite) Request

The upwrite request is used to save all the lines above (but not including) the current line in the segment specified by the user. The lines written out are deleted from the edit buffers and thus are no longer available for editing. They will replace the previous contents of path. The pathname specified by path can be either an absolute or a relative pathname. To invoke the request the user types:

```
! upwrite path
```

If path is not specified, the pathname given in the invocation of edm is used by default. If a pathname is given neither in this request nor in the invocation of edm, an error message is printed and edm looks for another request.

INDEX

HONEYWELL INFORMATION SYSTEMS
Technical Publications Remarks Form

CUT ALONG LINE

TITLE: SERIES 60 (LEVEL 68) MULTICS FAST SUBSYSTEM REFERENCE MANUAL ADDENDUM A

ORDER NO. AU25-01A

DATED DECEMBER 1979

ERRORS IN PUBLICATION

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Your comments will be investigated by appropriate technical personnel and action will be taken as required. Receipt of all forms will be acknowledged; however, if you require a detailed reply, check here. ☐

FROM: NAME _____ DATE _____

TITLE _____

COMPANY _____

ADDRESS _____

_____

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

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 39531 WALTHAM MA02154

POSTAGE WILL BE PAID BY ADDRESSEE

HONEYWELL INFORMATION SYSTEMS
200 SMITH STREET
WALTHAM, MA 02154

ATTN: PUBLICATIONS, MS486

FOLD ALONG LINE

CUT 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 Australia: 124 Walker Street, North Sydney, N.S.W. 2060
In Mexico: Avenida Nuevo Leon 250, Mexico 11, D.F.

24928, 5C1079, Printed in U.S.A.

AU25-01