Honeywell

USERS' GUIDE

MULTICS

SOFTWARE

Honeywell

USERS' GUIDE

MULTICS

SUBJECT:

Basic Introduction to Multics, Intended as a Guide for New Users.

SPECIAL INSTRUCTIONS:

For a more complete description on using the Multics System, refer to the *Multics Programmers' Manual*:

Introduction (Order Number AG90)
Reference Guide (Order Number AG91)
Commands and Active Functions (Order Number AG92)
Subroutines (Order Number AG93)
Subsystem Writers' Guide (Order Number AK92)

SOFTWARE SUPPORTED:

Multics Software Release 1.0

DATE:

November 1973

ORDER NUMBER:

AL40, Rev. 0

PREFACE

The purpose of this manual is to provide programmers and other users with a basic introduction to Multics use, a practice workbook that guides the new user through his first sessions at the terminal. The facilities described have been chosen either because they are immediately useful to the new user or because they are representative of the system as a whole.

The information presented here is a subset of that contained in the primary Multics reference document, the _Multics Programmers' Manual_ (MPM). The MPM should be used as a reference to Multics once the user has become familiar with this introductory guide.

Throughout this manual, references are frequently made to the MPM. For convenience, these references will be as follows:

Document	Referred To In Text As
Multics Programmers' Manual--Introduction (Order Number AG90)	MPM Introduction
Multics Programmers' Manual--Reference Guide (Order Number AG91)	MPM Reference Guide
Multics Programmers' Manual--Commands and Active Functions (Order Number AG92)	MPM Commands
Multics Programmers' Manual--Subroutines (Order Number AG93)	MPM Subroutines
Multics Programmers' Manual--Subsystem Writers' Guide (Order Number AK92)	MPM Subsystem Writers' Guide

CONTENTS

SECTION I

INTRODUCTION

Multics (Multiplexed Information and Computing Service) is a general purpose computer system developed at the Massachusetts Institute of Technology in cooperation with Honeywell Information Systems Inc. and other organizations. The Multics System multiplexes a central computer among the jobs of many users, each of whom accesses Multics from a terminal. With Multics, the user has facilities that allow him to edit, compile, debug, and run programs in one continuous, interactive session. Each Multics user can structure information, manipulate it, and simultaneously share it with other users.

This manual helps the new user become familiar with the Multics System. Basic Multics concepts, such as the storage system and access control, are described briefly. However, this manual offers a limited discussion of Multics concepts. Instead, it focuses on those facilities that any user needs daily, regardless of the nature of his terminal work.

Thus, the first topic in this document is the procedure by which the new user enters and exits the system plus a brief explanation of some terminal and Multics conventions (Section II). Next, the user is introduced to the Multics environment (Section III), that is, the storage system, naming conventions, and appropriate terminology. Once the user is familiar with the environment, the Multics commands are described (Section IV). The Multics System supports user-defined commands as well as system commands. Both types of commands and various command conventions are included in the commands description. As a special aid for the new user, certain frequently called system commands are also described (Section V).

To get his data on the system, modify the data, and save it, the user can call one of the Multics editors. One Multics context editor, edm, is very easy to use and gives the beginner the ability to manipulate his data through many simple requests (Section VI).

Of particular interest to the programmer is the description of programming in the Multics environment. While this manual does not teach programming, it does describe the ease with which programming can be accomplished on Multics (Section VII). The programmer can write his source program, edit it, compile it, debug and run the object program--a portion at a time if he chooses--and do it all online.

After describing the programming environment, the manual presents a brief discussion of the Multics access control concepts (Section VIII). On Multics, each user controls the access that other users have to programs and data he creates. Different access modes may be assigned to different users of the same program. The user who creates the program can set, change, and delete access rights just by invoking simple system commands. (Examples of these commands are also given in Section VIII.)

Another important, useful feature of Multics is online communication (Section IX). Through various commands, users can communicate with one another instantaneously, send mail, or even check to see who else is online at the moment.

The absentee use of Multics is briefly described in Section X. This facility is similar to batch processing on conventional systems.

The final section of this manual (Section XI) serves two purposes: it suggests further information to pursue for the novice, and it puts this manual in perspective with respect to the Multics System. The reader should understand that the material presented in this document represents only a fraction of the Multics System capabilities. However, this material should provide the new user with a challenge for two or three brief terminal sessions, after which Section XI and Appendices B and C will help to suggest further information to explore.

The new user will find Appendix A particularly helpful. It is a glossary of basic Multics terminology.

SECTION II

HOW TO ACCESS THE MULTICS SYSTEM

Before a user can gain access to Multics, he must be registered on the system by the site system administrator and allowed access to a particular project by that project's administrator. The system administrator assigns each user a unique two-part identification, consisting of a person identification (called a personid) and a project identification (called a projectid). The personid is generally a variation of the user's surname; the projectid is an arbitrary name for a project that is registered on the system. In addition, the system administrator assigns a special password for each user. For example, if Tom Smith were a new user, the system administrator could assign TSmith as his personid and ProjA as his projectid.

Notice that the personid (TSmith) contains capital letters. The Multics System distinguishes between uppercase and lowercase characters; if the exact capitalization is not used, the entry is misspelled and therefore not recognized by the system.

LOG-IN PROCEDURE

After the user has dialed the appropriate telephone number and a connection has been established between Multics and the user's terminal, Multics prints a message giving the number of the current system, the location of the system, the actual number of users logged in, and the number of users the system is currently accepting.

```
     Multics XX-x: PCO, Phoenix, Az.
     Load = 26.0 out of 100.0 units: users = 26
```

At this point, the user issues the login command and his personid, separated by a blank.

```
  !  login TSmith
     Password:
```

NOTE: Throughout this manual, the exclamation mark (!) is printed at the beginning of every line typed by the user. This is done only to distinguish user entries from system-generated printouts; the user should not actually begin his entries with an exclamation mark.

Also, a "carriage return" (moving the typing mechanism to the first column of the next line) is implied at the end of every user-typed line. See the glossary under "carriage return" to avoid confusion.

Multics then requests the user's password. Depending on the user's terminal, the printing of the password is either suppressed or hidden in a string of cover-up characters typed by the system. It is essential that the user keep his password secret to prevent unauthorized use of his programs and data and his account. If the user feels that his password has been compromised, he should notify his project administrator and immediately change his password. If the user ever forgets his password, he must notify the system administrator and request a new password. (Once a password is registered on the system, it is encoded and cannot be decoded by anyone, including the system administrator.)

If the user makes an error during the log-in procedure, the system informs him of it and asks him to try again.

```
     Login incorrect
     Please try again or type "help" for instructions.
  !  login TSmith
     Password:
  !
```

(The help command referred to in the system-generated printout is described in Section V.)

Each Multics installation sets its own limit on how many attempts a user may make to log in before the system automatically disconnects the line to the terminal.

```
     Login incorrect
     hangup
```

Project administrators may interpose another authentication procedure after the user types his password. The format of this procedure is determined by the individual project administrator.

After the user has successfully typed his password, the system responds with information regarding the user's last login.

```
     TSmith ProjA logged in 07/03/73 0937.5 mst Tue from terminal "234"
     Last login 07/02/73 1359.8 mst Mon from terminal "234"
```

The log-in statistics can be used to detect unauthorized use of the user's name and password on previous logins, since the user knows when he was last logged in. In addition, the log-in statistics inform the user of unsuccessful attempts to gain access to the system through his password. (Typing errors made by the user also count as unsuccessful attempts; see Section V for how to correct typing errors.)

```
  !  login TSmith
     Password:
  !
     Your password has been given incorrectly 2 times since last correct use.
     TSmith ProjA logged in 07/03/73 0937.5 mst Tue from terminal "234"
     Last login 07/02/73 1359.8 mst Mon from terminal "234"
```

These statistics are followed by the "message of the day." This message is a convenient way to tell all system users important news, including information on new commands and latest documentation.

```
The system will not be shut down on July 4.
Revision 1 of the Multics Programmers' Manual--Introduction is now
available--Order No. AG90.
```

The last line of system-generated printout in the log-in sequence is the ready message. This message is printed to indicate that Multics is at command level and ready to receive the next command. The ready message consists of the letter "r" followed by the time of day and three numbers that reflect system resource usage.

```
r 0937    1.314  1.332  30
```

These usage numbers identify virtual CPU time, which is the actual CPU time consumed by the user since the last ready message minus some supervisor execution time; memory units, which is an approximation of the amount of memory he has used since his last ready message; and the actual number of pages of information brought into main memory from secondary storage since the last ready message. For more information about the ready message, refer to the MPM Commands.

The complete log-in sequence for Tom Smith, assuming no one has attempted to use his password since his last login, would be:

```
!  login TSmith
   Password:
!
   TSmith ProjA logged in 07/03/73 0937.5 mst Tue from terminal "234"
   Last login 07/02/73 1359.8 mst Mon from terminal "234"
   The system will not be shut down on July 4.
   Revision 1 of the Multics Programmers' Manual--Introduction is now
   available--Order No. AG90.
   r 0937    1.314  1.332  30
```

Under certain circumstances, the user may be denied access to the system even though he has correctly logged in. For example, the system administrator may not yet have registered the user, the user may have exceeded the resource limits set for him by the project administrator, or the system may temporarily be full. In any case, if the user cannot get on the system, he receives a message from Multics telling him the reason he cannot log in and what steps, if any, he should take.

LOG-OUT PROCEDURE

When the user has completed his work, he breaks the connection between his terminal and the Multics System by issuing the logout command. The system responds by printing the identification of the user, the date and time of the logout, and the total CPU time and memory units used.

```
!  logout
   logout TSmith ProjA logged out 07/03/73 1249.4 mst Tue
   CPU usage 17 sec, memory usage 103.1 units.
   hangup
```

Some projects have a log-in time limit after which a user is subject to an automatic logout. In such cases, Multics prints a warning several minutes before a session is automatically terminated. Unless the user is automatically logged out by the system, he should always log out before leaving the terminal, to avoid wasting computer time and preventing others from logging in.

SECTION III

MULTICS ENVIRONMENT

One major component of the Multics environment, the virtual memory, allows the user to forget about physical storage of information; he does not need to be concerned with, or even aware of, where his information is within the system or on what device it resides. However, the new user does need to have a basic grasp of another major component of the Multics environment, the storage system, before he can begin to understand the system environment.

STORAGE SYSTEM

One good way to visualize the storage system is to consider it a "tree-structured" hierarchy of directory segments. The basic unit of information within the storage system is the segment; it may contain a collection of program instructions or data, or it may be empty (a null segment). Some segments serve as catalogs of other segments beneath them in the tree structure, listing the attributes of the subordinate segments; these cataloging segments are called directory segments. All the other segments are called nondirectory segments. However, by convention directory segments are called simply "directories"; nondirectory segments, simply "segments."

At the beginning of the tree is the root directory; all other directories and segments emanate from the root directory. For example, Figure 3-1 shows user Tom Smith and his project, ProjA, in relation to the root. (Directories are represented by rectangles and segments by circles.) Notice the two directories immediately under the root (library and udd). The library directory is a catalog of all the system commands. The udd (user_directory_directory) is a catalog of project directories. It contains one directory entry for each project on the system. Likewise, each project directory normally lists one directory for each user on that project.

NAMING CONVENTIONS

The actual name of any segment or directory reflects its position in the hierarchy in relation to the root directory. This name, called the pathname, shows the "path" from the root directory to the specific segment or directory. Each name between the root and the specific segment or directory indicates another level in the directory hierarchy. To refer to a particular segment or directory, the user must list these names in the proper order (i.e., beginning with the root and coming down) and must include a greater-than symbol between each name. The greater-than symbol (>) is used in Multics to denote hierarchy levels.

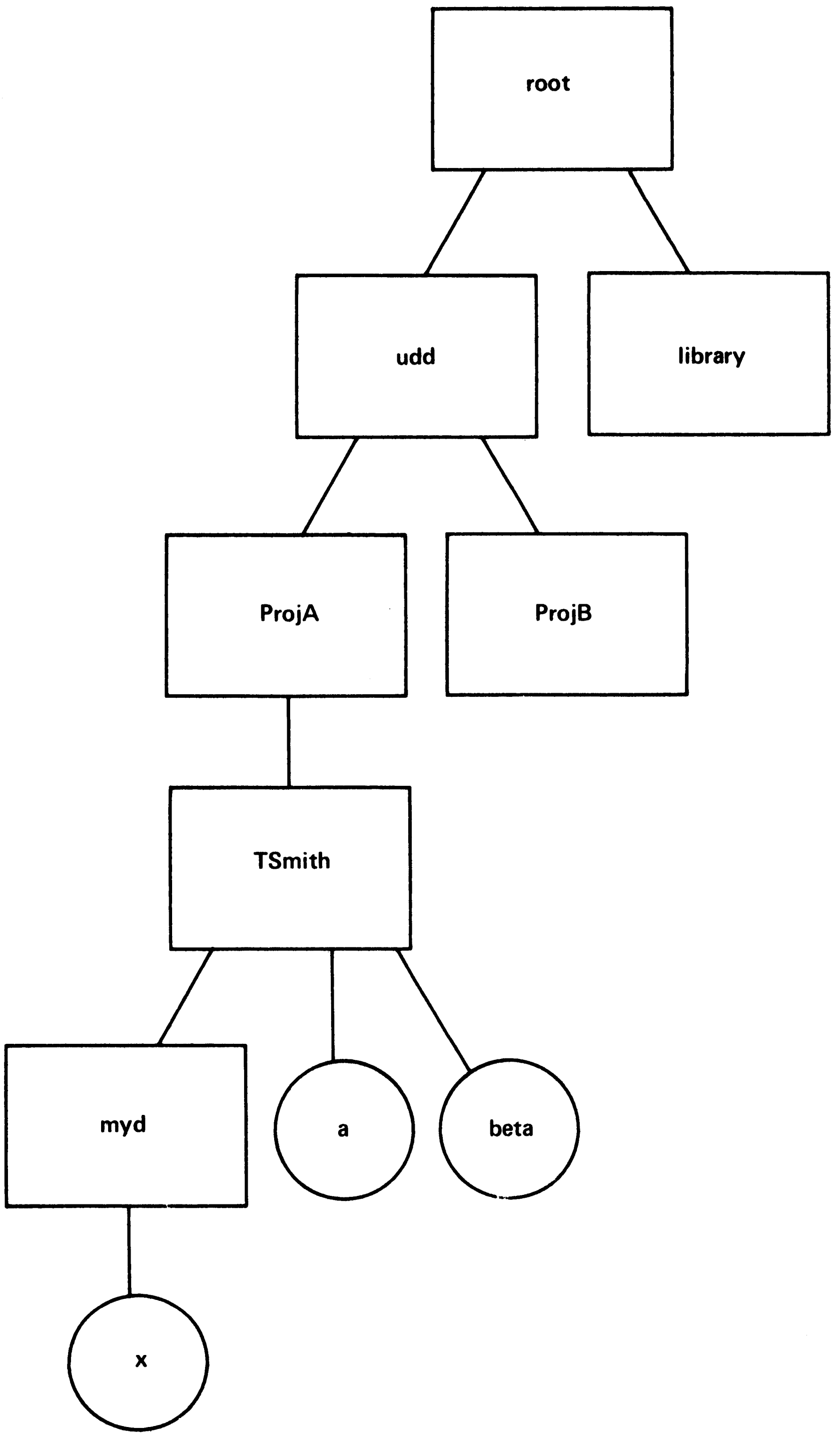

Figure 3-1. Hierarchical Storage System

The pathname for segment x in Tom Smith's myd directory is:

```
>udd>ProjA>TSmith>myd>x
```

By convention, the word "root" is omitted since it would be the first entry in every pathname. The pathname in the above example is called an absolute pathname because it identifies the "absolute" or complete path between the root and the specific segment. For the user's convenience, Multics also accepts a shortened version of the pathname, called the relative pathname, that identifies the specific segment in relation to the current working directory.

The working directory is simply the directory in which the user is currently working; it identifies his current location within the storage system (see also the following paragraph on directory concepts). Because the system keeps track of a user's working directory, the user needs to identify only the names between his working directory and the specific segment. Any name the user types that does not begin with the greater-than symbol is considered relative to his working directory. Thus, the relative pathname for segment x when TSmith is the working directory is:

```
myd>x
```

and the relative pathname for segment x when myd is the working directory is simply:

```
x
```

Each individual name in the pathname is called an entryname. An entryname is a user-assigned identifier, from one to 32 characters long, chosen from the full ASCII[1] character set (excluding the greater-than and less-than symbols). The user should avoid the use of a space as part of an entryname; it is permitted but cumbersome because the command language uses spaces to delimit command names and arguments. Also, use of other special symbols such as the asterisk (*) and equals (=) characters is not recommended; these symbols have special meanings in many Multics standard commands. (Refer to Appendix B for information on these special meanings.)

Entrynames must be unique within any one directory. For example, Tom Smith can use "test" as an entryname in directory TSmith and also in directory myd; furthermore, he could also use "test" as an entryname in a directory named "test." However, he cannot use "test" as the name for two different segments in the same directory. Also, it is permissible--and very convenient--to assign more than one entryname to any segment or directory. For example, the entryname "test" would be easier to type than "part1.test.new_compiler" any time Tom Smith wanted to work with that particular segment.

[1]American Standard Code for Information Interchange

WORKING DIRECTORY CONCEPT

Each user on Multics functions as though he performs his work from a particular location within the Multics storage system--his working directory.

Working Directory

The working directory is just that--the directory in which the user is currently doing his work. The main purpose of the working directory is convenience; the user does not have to take the time to type in absolute pathnames because the system assumes that any name he types that does not begin with the greater-than symbol is relative to the current working directory. Thus, the user can type the shorter relative pathname, and the system, knowing the current working directory, supplies the rest of the pathname.

The user can change his working directory simply by invoking a standard Multics command (see Section V). Then he can enter pathnames relative to the new working directory, and the system again supplies the rest of the pathname.

Initial Working Directory

Whenever a user logs in, he logs into a particular directory within the storage system; that is, the system sets his working directory for him. This initial working directory is known as the home directory. The system "remembers" the pathname of each user's home directory and automatically assigns the user to that directory when he logs in. Generally, the home directory is:

```
>udd>projectid>personid
```

For example, Tom Smith's home directory would be:

```
>udd>ProjA>TSmith
```

Although the user can change his working directory, he cannot change his home directory. No matter what the identity of his working directory when he logs out, his working directory when he logs in is always his home directory.

SECTION IV

COMMANDS

A command is a procedure that is called from the terminal by typing the name of the command (or its abbreviation) followed by character string arguments. These arguments specify names of data segments containing information to be acted upon by the command or special kinds of information needed to execute the command. For example, the command

```
print alpha
```

causes Multics to locate segment alpha within the user's working directory and print it on the terminal.

SYSTEM COMMANDS

System commands are kept in directories called system libraries and are supplied with the Multics System. The typical user does not have permission to rewrite these commands, but he does have full permission to use them. System commands are well engineered and tested and are supplied with self-explanatory error messages. If the user makes a typing error or types a command that does not exist, an explanatory message is typed on the terminal. For example, if the user wants to invoke the command, print alpha, and instead types

```
! primt alpha
```

the system responds with an error message and a ready message (returns to command level)

```
Segment primt not found.
r 0937    .144  2.402  55
```

USER-WRITTEN COMMANDS

User-written commands are cataloged in directories of the user's choice. These commands may have the same names as system commands; they are distinguished from system commands by their location.

Multics searches in various locations (user directories, system libraries, etc.) in a particular order to find the requested command. The user can alter that order by using a system command to change the search rules (see Section XI). By redefining the search rules, the user can determine whether a user-written command or the system command is to be used where both have identical names. For example, when a user types a command at his terminal, like print alpha, the system first interrogates the user's own directory to see if the "print" program exists. If so, that version is executed. If not, the system searches the system- and user-supplied directories (according to the rules specified by the user) for the "print" program. If the "print" program is still not found, the system types an error message and returns to command level.

STOPPING DURING COMMAND EXECUTION

If the user wants to halt program execution at any time, he can do so by issuing a quit signal. The user invokes the quit signal by pressing the proper key on his terminal (e.g., ATTN, BRK, INTRPT, INTERRUPT). As soon as the system receives this signal, Multics stops executing his program and types a ready message.

For example, when the user issues the print command, he may not need to see the entire segment. So as soon as the system prints the information he needs, he issues a quit signal. The quit signal causes Multics to stop printing the segment and issue a ready message. The system is now at command level, ready to accept the next command.

If the user wishes to continue the work interrupted by the quit signal, he can issue either the start or the program_interrupt command. The start command resumes execution of the original program from the point of interruption. The program_interrupt command resumes execution of the original program from a known, predetermined reentry point. If the user does not want to continue the interrupted work, he should issue the release command before he issues any other commands. The release command releases the work interrupted (and held) by the quit signal. For more information on these commands and their relation to the quit signal, see the MPM Commands.

COMMAND CONVENTIONS

The general format of Multics commands is:

```
command argument1 argument2 ... argumentn
```

Most Multics commands have various arguments that allow the user to modify command execution to suit his needs. However, the new user does not have to know any argument except the pathname for many Multics commands. By using the simplified command line format, command pathname, the new user can effectively use the Multics System.

For the user's convenience, a brief description of storage system and command conventions is given in Appendix B. Also, for a description of the command language environment, refer to the MPM Reference Guide.

SECTION V

SAMPLE COMMAND EXECUTION

This section first shows the procedure for correcting typing errors and then illustrates the use of certain, frequently called Multics commands. Since edm, access control, and the online communication commands are fully described elsewhere (Sections VI, VIII, and IX respectively), they are not included here.

CORRECTING TYPING ERRORS

There are two special symbols for correcting typing errors, the character delete and the line delete. These symbols may vary, depending on the type of terminal; but generally the number sign (#) is the character-delete symbol, and the commercial at sign (@) is the line-delete symbol.

The character-delete symbol "erases" one previously typed character, space, or tab when typed directly after the error. The line-delete symbol "erases" every character previously typed on the line, including spaces and tabs. Examples of both symbols are given in the login command lines below. Each line is interpreted by Multics as--login TSmith.

```
! login TSM#mith

! logen T####in TSmith

! logen TSmit@login TSmith

! kigum T@loge#in TSmith
```

PRINT WORKING DIRECTORY (print_wdir) COMMAND

The print working directory command (invoked by typing print_wdir or pwd) requests that the system print the name of the working directory. Multics responds by typing the absolute pathname of the user's working directory.

```
! pwd
  >udd>ProjA>TSmith
  r 0938    1.347  2.315  41
```

CHANGE WORKING DIRECTORY (change_wdir) COMMAND

The change working directory command (invoked by typing change_wdir or cwd) redefines the working directory. To change his working directory, the user types the cwd command followed by the pathname of the directory he wishes to redefine as his working directory. The following command changes the working directory to JDoe.

```
!   cwd >udd>ProjB>JDoe
    r 0938    .972  1.731  25
```

To revert to the initial working directory, the user types the cwd command without an argument.

```
!   cwd
    r 0939    1.024  1.378  35
```

The cwd command allows the user to manipulate his own position within the storage system hierarchy with respect to the segments he wishes to use. However, the user must remember that it will not help to change working directories if he does not have access permission to use segments cataloged in that directory.

LIST (list) COMMAND

The list command (invoked by typing list or ls) prints out a list of all the segments in a directory. If the user issues the list command with no arguments, the working directory is assumed. The command prints information about the number of segments and records in the directory, access attributes, and number of records for each segment. Segments most recently created are at the top of the list.

```
!   ls

    Segments = 21, Records = 46.

    r w    7  index
    rew    2  alpha
    re     1  beta
    .
    .
    .
    r w    4  gamma

    r 0939    1.866  2.084  41
```

PRINT (print) COMMAND

The print command (invoked by typing print or pr) prints the contents of an ASCII segment. The segment name may be either an absolute or relative pathname.

```
  !  pr sectionV
                        sectionV     07/03/73  0940.2 mst Tue

          This section first shows the procedure for correcting typing
     errors and then
     illustrates the use of certain, frequently called
     Multics commands.  Since edm, access control, and the
     online communication commands are fully described elsewhere
     .
     .
     .
     .

     r 9040     1.245  1.921  53
```

HELP (help) COMMAND

The help command (invoked by typing help) prints information about commands, the current system, etc. When the help command is invoked, it prints out the specified information segment a portion at a time, identifying the number of lines that follow and giving the user the option to continue. For example, if the user wants information about the arguments and options available with the list command, he types:

```
  !  help ls
     (6 lines follow)
     11/17/71


     The "list" command lists the contents of a directory.
     To list segment names only, use "listnames" ("ln").
     To list directory totals only, use "listotals" ("lt").


     Rest of segment has 14 lines.  More help?
```

At this point, the user types either "yes" or "no" depending on whether or not he wants more information about the command.

To print the names of available text segments in the help information segments, type:

```
  !  help help
```

SECTION VI

MULTICS EDITOR

The edm command, which is a simple Multics context editor, 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. This list does not include all of the edm requests; it identifies only those requests that the new user will need as he begins using Multics. For a complete listing and description of all the edm requests, see the MPM Commands.

-	backup
=	print current line number
,	comment mode

.	mode change
b	bottom
c	change
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 entry 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
!  c/m/n/
   orange
!  b
   Input.
!  blue
```

Change (c) Request

The change 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:

```
!  c4/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.

NOTE: For compatibility with the qedx editor, the substitute (s) request may be used in place of the change (c) request to accomplish the same results.

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 throughly 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 is identical to the change (c) request.

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 <u>and</u> 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.

SECTION VII

PROGRAMMING ON MULTICS

This manual is not intended to offer instruction in programming; instead it describes programming within the Multics environment. The basic steps are still the same: write, compile, execute, and debug.

However, on Multics the user can do all four steps online in one terminal session. He has source language debugging capabilities that allow him to execute and test only certain portions of a program if he wishes. The Multics virtual memory and storage system eliminate the file input/output normally required to manage the transfer of information to and from secondary storage; physical movement of data from primary memory ("core") to secondary storage and back is wholly automatic and of no concern to the programmer. In addition, dynamic linking and the organization of the storage system eliminate the need for extensive software management, since the latest copy of every program is immediately accessible by name from the terminal or from a program. This dynamic linking capability eliminates the need for a complicated job control language for retrieving, prelinking, and executing programs and for defining and locating input/output files.

WRITING A SOURCE PROGRAM

To write a source program, the user invokes an editor that allows him to input--and then edit--his work. On Multics, the user can write his source program in a variety of languages such as PL/I, FORTRAN, or BASIC, or even a language he himself has devised.

The following example shows a PL/I source program entered on the system with the edm editor.

Terminal Interaction	Explanation
`! edm prog.pl1`	User types the edm command and the name of the segment.

`Segment not found.`
`Input.`
	edm searches the working directory for prog.pl1. It is not found so edm creates it and switches to input mode. (If prog.pl1 already existed, the user would be put in edit mode.)

`! dcl a fixed bin(17),`
`!     b char(*,`
`!     c external entry;`
	User input. The second line is missing a right parenthesis.

```
!  .                                User changes to edit mode.
   Edit.

!  l char                           User issues locate request to find char.

       b char(*,                    edm prints line containing char.

!  c/,/),/                          User issues change request to enter the
                                    missing parenthesis.

       b char(*),                   edm prints out changed line.

!  n                                User issues request to print next line.

       c external entry;            edm prints next line.

!  w                                User writes prog.pl1 into his working
                                    directory to save it for later use.

!  q                                User quits edm.
```

COMPILING A SOURCE SEGMENT

The basic (bs), fortran (ft), and pl1 commands compile source segments written in the respective languages. Source segment names must include the name of the language as the last component, separated by a period.

The mandatory last component for source programs written in PL/I is pl1 (e.g., alpha.pl1 or alpha.beta.gamma.pl1). Although pl1 must be used as the last component in the source program name, it need not be used in the pl1 command, which compiles the source program. The last component is understood as being implied by each of the language processors.

For example, to compile the PL/I source program, prog.pl1, the user invokes the pl1 command by typing:

```
pl1 prog -table

   or

pl1 prog.pl1 -table
```

The optional argument added here, -table, is one of the many options accepted by the pl1 command. The table option produces a symbol table that is valuable for use with the Multics debugging facility.

EXECUTING A PROGRAM

To run an object segment, the user simply types its name (either relative or absolute pathname). To run the compiled version of source program prog.pl1, the user types:

```
prog
```

So far as Multics is concerned, each program may be a command.

Programs may reference Multics subroutines that allow programs to accept online data. The user simply supplies the necessary data at the terminal as the running program requests it by typing in the required number of arguments or responses. Or, the user may input arguments as he types the name of the program, the way he does for a command. Multics also provides subroutines to print results of program execution on the terminal.

DEBUGGING A PROGRAM

Multics permits users to run a part of a program, temporarily halt its running, debug that portion of the program needing changes, and resume running the program.

Most Multics compilers have the ability to print a list of errors when they compile a source segment. For example, the list might indicate incorrect syntax in the source segment. This list of errors may be graded by severity; the user may judge whether he wishes to continue compilation or halt it by issuing a quit signal. Severe errors automatically cause compilation to cease. The compiler prints an error message; and the system returns to command level and prints a ready message.

The debug (db) command accesses a facility that allows users to look at and/or modify data or code online. The user may set "breakpoints" within the program, run the program, and cause the program to halt at the breakpoints. The user may then test the program using various debug requests, and use a text editor to modify the source program. This debugging facility allows symbolic references, permitting the user to depart from machine-oriented debugging techniques.

For more information on the debug command, refer to the MPM Commands. Also, the MPM Introduction provides an extensive example on the use of the debug command under "Programming in the Multics Environment."

SAMPLE PROGRAM

This paragraph shows the terminal interaction as a user logs in and writes, compiles, and executes a short program. The program, named add, is written in PL/I using edm. The add program accepts two integers (online) and prints the sum of the two integers on the terminal.

```
!  login TSmith
   Password:
!
   TSmith ProjA logged in 07/03/73 0937.5 mst Tue from terminal "234"
   Last login 07/02/73 1359.8 mst Mon from terminal "234"
   The system will not be shut down on July 4.
   Revision 1 of the Multics Programmers' Manual--Introduction is now
   available--Order No. AG90.
   r 0937    1.314  1.332  30

!  edm add.pl1
   Segment not found.
   Input.
!  add: proc ;
!  dcl (n1,n2,sum) fixed bin(17);
!  dcl (sysin, sysprint) file;
!  put list ("This program prints the sum of two user-supplied integers.");
!  put list ("Enter two integers separated by a comma.");
!  put skip;
!  get list (n1,n2);
!  sum = n1 + n2;
!  put skip;
!  put list ("The sum is:", sum);
!  put skip;
!  end;
!  .
   Edit.
!  w add.pl1
!  q
   r 0944    4.875  7.621  62
```

After typing in the source program, going to edit mode to write it, and quitting edm, the user is ready to compile his program. Notice that the program name (add.pl1) includes the language name as the last component. The language name also identifies the proper command to invoke for the compilation. Thus to compile the add.pl1 program, the user types:

```
!  pl1 add
   PL/I, Version 2
   r 0945    1.635  28.516  383
```

Once the program is compiled, the user, and any other users to whom he gives proper access, can execute the program by typing:

```
!  add
   This program prints the sum of two user-supplied integers.
   Enter two integers separated by a comma.
!  86425,999

   The sum is:              87424
   r 0946    .191  1.436  57
```

SECTION VIII

ACCESS CONTROL

On the Multics System, the user is able to share as much or as little of his work with as many other users as he desires. The checking done by the hardware on each memory reference ensures that the access privileges described by the user for each of his segments are enforced. This kind of privacy and security gives Multics users great flexibility in the kinds of data they may put on the system. For example, if Tom Smith were the head of a personnel department, he could put the names and addresses, salaries, education, etc. of all the company's employees online. He could then set different access rights on each of the segments. For example, he could assign read and write access to only himself on the segment containing salary information. He would not allow anyone else in the department to have any access to the salary segment. On the segment containing the names and addresses of all personnel, he could assign read and write access to himself and his assistant and only read access to the rest of his department. Multics allows the user to give different access rights to different users of the same segment.

ACCESS CONTROL LIST

The access rights for each segment are described in an access control list (ACL). Each segment has its own ACL; it contains the identification of users permitted (or specifically denied) access to the segment plus a description of the type of access allowed.

The user identification in the ACL consists of a three-component name: personid, projectid, and an instance tag, separated by periods. (The system assigns the instance tag 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 attributes of segments, directories, and links contained in the directory can be obtained
modify	(m)	the attributes of existing segments, directories, and links contained in the directory can be changed or deleted
append	(a)	new segments, directories, and links can be created in the directory
null	(n)	access to the directory is denied

The user generally assigns combinations of access modes to his segments and directories. Useful access mode assignments for segments and directories are:

Segments	Directories
r	s
re	sm
rw	sa
rew	sma
null	null

The user specifies one of the above access mode assignments for the persons and/or projects he wishes; he uses one command in specifying access to his directories and/or segments. Once specified, the access is not "frozen"; the user may change it at will just by issuing the command again, specifying different modes, persons, or projects.

SETTING ACCESS

The command the user invokes to set the ACL, setacl, either adds an entry to the ACL or modifies an existing entry. The setacl command, which may be abbreviated sa, has the general format:

```
     sa  pathname  accessmode(s)  useridentification
```

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

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

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 personid (JDoe) in the above command lines tells the 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 ACL entry Tom gave, JDoe.*.*, matches:

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

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

LISTING ACCESS

To check the ACL of a segment, the user invokes the command that lists the ACL, listacl. The listacl command, which may be abbreviated la, has the general format:

```
la  pathname
```

As explained earlier, the system assumes that any pathname that does not begin with the greater-than symbol is relative to 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 the system processes will have the necessary access to perform the various backup, metering, and input/output functions.

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

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

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

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

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

DELETING ACCESS

A third access control command, deleteacl, allows the user to delete ACL entries. This command, which may be abbreviated da, has the same general format and rules as the listacl 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       Gray.Merlin.*
   rw       Butler.Merlin.*
   rw       Jones.*.*
   re       JDoe.*.*
   rw       *.SysDaemon.*
   r        *.*.*
```

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

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

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

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

On Multics, changes in access rights occur instantaneously. If both Tom and Jane are online at the same time and she tries to access one of his segments and finds that she does not have the proper access, she can send him a message (see Section IX concerning online communication), asking him to give her proper access to the segment. He can then issue the appropriate arguments to the setacl command, and she immediately has access to the segment. Access rights are revoked just as rapidly. If Jane has access to a segment of Tom's, and he changes the access while she is using the segment, the system prints out a message telling her that she has incorrect access to the segment and returns her to command level.

SECTION IX

ONLINE COMMUNICATION WITH OTHER USERS

The Multics System offers several commands that enable users to communicate with one another online. Such commands are extremely useful; for example, a user may require immediate access to another user's data, or a user may need to request an increase in his quota from his project administrator.

mail COMMAND

The mail command (abbreviated ml) is used to send mail to another user or print mail sent by another user. The mail is stored in the user's home directory in a segment with the entryname, mailbox. Thus, this segment has the absolute pathname:

```
>udd>projectid>personid>mailbox
```

In order to receive mail, the user must create the mailbox segment and set the access control list (ACL) to rw for all users.

```
!  create mailbox
   r 1145    1.327  1.944  57

!  setacl mailbox rw *.*.*
   r 1145    1.856  2.361  61
```

To read his mail, the user types:

```
!  mail
```

The system first tells the user how many messages are in his mailbox and the total number of lines in the mailbox. Then the system prints all of the messages and asks the user if he wants to delete these messages. If the user answers yes, the messages are deleted; if he answers no, they are saved.

To send mail, the user types the mail command with the proper arguments.

```
mail message personid projectid
```

The message argument may be either the pathname of a segment or an asterisk (*). Generally, the user types an asterisk for the message argument. The system responds by printing "Input." The user then types his message, ending it by a line containing only a period (.). For example, if Tom Smith wants to send Jane Doe mail, he types:

```
!  mail * JDoe ProjB
   Input
!  Dear Jane,
!  Thank you for the draft copies of
!  the new manual.  I promise to return
!  them next week.
!                           Tom
!  .
   r 1004    .741  3.386  99
```

If the information the user wishes to send is contained in a segment, he types the pathname of the segment for the message argument. The content of the segment is then placed in the receiving user's mailbox.

When the system sends the mail, it supplies a header identifying the sender and the date and time he sent the message. It then copies the mail with the header into the proper mailbox segment--in this case, >udd>ProjB>JDoe>mailbox. Thus, when Jane checks her mailbox, by issuing the mail command, Tom's message would appear on her terminal as:

```
!  mail
   1 message, 8 lines.
   From TSmith.ProjA  07/09/73  1004.6

   Dear Jane,
   Thank you for the draft copies of
   the new manual.  I promise to return
   them next week.
                            Tom

   mail: Delete?
```

MESSAGE FACILITY

The message facility permits online communication between users on different terminals. The system puts messages from other users in a segment called the message segment and prints them out at the option of the user, i.e., either immediately or on command. Even if the user who is to receive the message is not logged in or is deferring messages, messages can still be sent. If this happens, the user who is sending the message is notified that the message cannot be received online at this time, and the message is stored in the receiving user's message segment.

If the user wants the ability to receive messages, he must invoke the accept_messages command (or its abbreviation, am). This command causes the system to create the message segment, if one does not already exist, having the absolute pathname:

```
>udd>projectid>personid>personid.con_msgs
```

If Tom Smith wants to use the message facility, he types:

```
! accept_messages
  >udd>ProjA>TSmith>TSmith.con_msgs has been created.
```

Once the accept_messages command is invoked, the user receives--instantaneously--any messages sent to him during that terminal session. If a message is sent after he logs out, it is saved in the message segment. In order to receive the messages saved in the message segment, the user must issue the print_messages command (or its abbreviation, pm).

```
! print_messages
```

(To avoid the necessity of issuing the accept_messages and print_messages commands at the beginning of each terminal session, refer to the discussion of the start_up.ec segment under the "exec_com command" description in Section XI.)

When the user wants to send a message, he must invoke the send_message command (or its abbreviation, sm) with the proper arguments.

```
send_message personid projectid message
```

If Tom Smith wants to send a message to Jane Doe, he types:

```
! send_message JDoe ProjB Gave you access to xsolve--you can use it now.
```

If the message argument is missing, the system types the word "Input" and the user then types his message. Each line of his message is sent as soon as the user types the carriage return. To indicate that the message is complete, the sending user types a period (.) on a separate line.

Whenever a message is sent, the sending user is identified by his personid and his projectid. Tom's message would appear on Jane's terminal as:

```
from TSmith.ProjA: Gave you access to xsolve--you can use it now.
```

who COMMAND

Although the who command is not an online communication command in the same sense as mail, accept_messages, print_messages, and send_message, it is still an important communications tool. By invoking this command, the user learns the number, identification, and status of all users currently on the system. (It is possible for a user to prevent his name from being listed; to do this, the user should first see his project administrator.) Often, the user issues the who command to see if a particular user is online before he issues the send_message command.

The who command first prints out a header line, listing the system name, the total number of users, the current system load, and the maximum load. After this header, the command lists the name and project of each user.

To invoke the who command, the user types:

```
! who

  Multics XX-x, load 12.0/90.0; 12 users
  Absentee users 0/3

  IO.SysDaemon
  Backup.SysDaemon
  Metering.SysDaemon
  Dumper.SysDaemon
  JDoe.ProjB
  Rolf.Alpha
  RSmith.North
  TSmith.ProjA
  Green.Beta
  Richards.ABC
  Camp.Gamma
  Warren.XYZ

  r 2350    .668  1.036  74
```

The list of users in the above example is sorted according to log-in time. The user can specify certain options when he calls the who command and change the sort key, suppress the header, list only those users of a particular project, and various other things. For information about the control arguments and other arguments that can be used with the who command, refer to the MPM Commands.

SUMMARY

This section shows the new user enough information about five communication commands to enable him to use them. However, he should realize that the mail, accept_messages, print_messages, send_message, and who commands offer a variety of options to make online communication even more meaningful to any one particular user; that is, each user can choose those options that are most suitable to the type of online work he does.

For example, a user who is printing out a final draft of a document would certainly not want to receive a message from another user in the middle of his printout. Therefore, he may wish to defer messages while he is printing the draft; he still "receives" the messages, but they are saved in his message segment (with the absolute pathname >udd>projectid>personid>personid.con_msgs) until he asks for them.

To learn more about the various control arguments and options available with these communication commands, refer to the MPM Commands.

SECTION X

INTERACTIVE AND ABSENTEE USAGE

Although this manual deals exclusively with the interactive usage of Multics, the new user should be aware that there is another type of Multics usage--absentee.

Absentee usage (similar to batch processing in other systems) gives users the ability to execute large production runs without waiting at the terminal while the run is in progress. The user merely creates an absentee job and submits it for execution.

Absentee jobs are placed in a queue and run as background to the normal interactive work of the system. Because absentee jobs are usually deferred until the interactive load is light, the charges for absentee usage are substantially lower than the charges for interactive usage.

To create an absentee job, the user creates an absentee input segment that contains those commands he wants executed. The job control language for absentee usage is identical to the command language for interactive usage. Basically, an absentee job is merely a "planned" interactive terminal session; that is, the user anticipates any responses or commands he must give and puts all of this data into his absentee input segment.

For more information about absentee usage and a complete description of how to request an absentee job, see the enter_abs_request command in the MPM Commands.

SECTION XI

MULTICS FEATURES FOR ADVANCED USERS

Once the user becomes familiar with most of the basics, as described in earlier sections, he begins to take "shortcuts" by learning certain system features that he finds extremely useful in his particular type of work at the terminal. The purpose of this section is to enumerate and briefly describe several such features. These descriptions also identify appropriate reference materials so that interested users can easily add these features to their Multics repertoire. The fact that these features are part of the Multics System does not mean that they all need to be learned by any one user; each user learns the features he feels will be most helpful to him.

ABBREVIATION PROCESSOR

The abbreviation processor is a special command processor that is invoked for each command line *after* the user invokes the abbrev command. The user defines his own abbreviations for frequently used command lines or other strings. Then, after invoking the abbrev command, the abbreviation processor checks each command line for abbreviations; any abbreviations the user has defined are expanded by the abbreviation processor and then passed on to the Multics command processor.

Use of the abbrev command (refer to the MPM Commands) greatly simplifies the user's terminal work. For a special use of the abbrev command, see the exec_com command description in this section.

ACTIVE FUNCTIONS

An active function is a program that is invoked as part of a command line; the character-string result of the active function replaces its invocation in the command line. In other words, the active function part of the command line is executed immediately, the result placed in the command line, and *this* command line is passed on to the command processor for execution.

The active functions fall into seven operational groupings: logical, arithmetic, character string, segment name, date and time, question asking, and user parameter. Refer to the MPM Reference Guide (under "The Multics Command Language Environment") for a complete description.

By using active functions, the user is able to make many standard Multics commands conditional commands. For example, the user may want to set the length of his command line to x only if he is working on a particular type of terminal; or he may want to enter a certain absentee request only if today's date equals a particular number.

ADMINISTRATIVE FEATURES

Multics administration defines three levels of responsibility: system, project, and user. A system administrator allocates system resources among the projects; a project administrator allocates these resources among the users on his project; users can manage their own data through storage management and access controls.

All of the administrative operations can be performed while the system is running; desired actions take place immediately. Multics administrative operations cover the following areas:

Resource distribution
Accounting and billing operations
Usage control
Environment shaping
Access control and security

If Tom Smith is the project administrator for ProjA, he can determine the dollar limit that a particular ProjA user may incur in a single month. If this limit is exceeded, the user is automatically logged out; he cannot log in again until either the next month begins or until the limit is changed.

As project administrator, Tom can also determine several other items, including whether a user can preempt others, specify his home directory, or have primary or standby status. In fact, the project administrator can so control each user's environment that he can deny a user access to the full Multics System and instead provide the user with access to only those commands that he (as the project administrator) specifies. Such a user is said to have access to a limited service system.

For more information on the Multics administrative features, refer to the System Administrator's Manual (Order No. AK50) and the Project Administrator's Manual (Order No. AK51).

ARCHIVE SEGMENT

An archive segment is a single segment consisting of the contents of many different segments packed together. Once in an archive, the individual segments are called components of the archive segment. This packing, performed by invoking the archive command, reduces the user's storage load.

By invoking the archive command with different arguments, the user can manipulate the archive segment in a variety of ways. For example, he not only creates his archive; he also can get a table of contents that names each component in the archive, extract one or more components from the archive, update and replace one or more components, and delete individual components. For more information about the archive command and its use, refer to the MPM Commands.

BOUND SEGMENT

A bound segment is a single executable procedure segment ("object segment") made up of one or more separately compiled, executable procedure segments. Again, as with an archive segment, the user reduces his storage load by combining several segments. However, a bound segment also automatically prelinks all the internal intersegment references thereby reducing execution time.

The user creates a bound segment by invoking the bind command. The bind command also allows the user to update, list, and map the bound segment as well as manipulate the manner in which the various segments in the bound segment are to be called.

Those programs that the user calls frequently and that are interrelated (i.e., reference one another) should be bound to improve program efficiency. By putting such programs in a bound segment, the user saves money through decreased compute time and storage space and, at the same time, decreases his execution time.

For more information about the bind command, refer to the MPM Commands. Also, the MPM Subsystem Writers' Guide provides more information on the structure of bound segments.

exec_com COMMAND

The exec_com command permits the user to execute a series of commands specified in a segment. This segment must contain only command lines and control lines; also, it must have the letters ec as the last component of its name (e.g., test.ec or alpha_1.beta.ec). The exec_com command also allows the user to substitute special strings in the segment by giving certain arguments when he invokes the command.

The command lines in the segment can use any Multics command. The control lines are defined in the exec_com command description (refer to the MPM Commands).

Multics permits users to create a special exec_com segment that contains commands to be executed when the user logs in, before his process attempts to read from his terminal. In other words, he does not even need to invoke the exec_com command for this segment; it is automatically invoked for him as part of his log-in procedure. This segment must be named start_up.ec and must reside in the user's initial working directory. Some commands that a user typically includes in his start_up.ec are:

abbrev	so he doesn't need to remember to invoke it during each terminal session in order to use his abbreviations
accept_messages	so he can create the message segment and then receive online messages from other users
print_messages	so he can receive messages saved in the message segment

mail	so the system automatically prints the mail that has been sent to him since his last terminal session
print_motd	so the system keeps a record of the message of the day and prints only those portions that differ from the last message the user saw

Also, a user generally writes his start_up.ec so that certain lines of the segment are executed when he logs in as an interactive user and other lines are executed when he is logged in as an absentee user. For example, the accept_messages, print_messages, mail, and print_motd commands would not be used in a start_up.ec for an absentee user.

INPUT/OUTPUT SYSTEM

The Multics System contains a device-independent input/output system interface that programs can use. This interface allows interchangeable reading and writing via tapes, terminals, cards, printers, and storage system segments.

This generalized input/output means that segments and input/output devices are interchangeable since both are referenced by symbolic name. A user can place a series of commands in a segment, attach the segment to an input stream, and the system will process the user computation indicated by the commands as if input were from the terminal. An interface command that assigns input/output streams (the ioc command) is available to all users. Output also can be directed to a segment by issuing the file_output command. Moreover, the user can switch from input device to input segment and from output device to output segment in the same way that he can switch from device to device. In addition, Multics provides system commands for user output that automatically queue the specified segments for printing (the dprint command) or punching (the dpunch command).

The Multics input/output system was designed for flexibility. In fact, users can write their own input/output routines, which can be "plugged in" to this system.

For information about the various input/output commands, see the MPM Commands. A detailed description of the Multics input/output system is given in the MPM Reference Guide.

LINKING SEGMENTS

Multics allows a user to create a link to a segment anywhere in the storage system as long as he has the proper access to the directory in which the link is to be placed. The user invokes the link command to create a link (refer to the MPM Commands).

By creating a link, the user is able to reference another segment as though it were in the directory containing the link. In short, he has the use of this particular segment without actually having to make a copy of it. This linking feature allows users to share information easily and inexpensively.

qedx EDITOR

The qedx context editor can be used to create and edit ASCII segments in a manner similar to the edm editor (described in Section VI). However, qedx is a more powerful editor than edm. It honors global editing requests and supports a virtually unlimited number of buffers. In addition, the macro capabilities of qedx make it almost a programming language in itself.

At any one time, one of the qedx buffers is designated as the current buffer and all others are auxiliary buffers. The user can move information from one buffer to another, designate any buffer as the current buffer, and check the status of all the buffers. The user can place a frequently used editing sequence in one buffer and then through a special escape sequence invoke the contents of this buffer whenever necessary.

The user can place elaborate editor request sequences (called macros) into auxiliary buffers and then use the editor as a pseudoprogramming language. In a sense the macro is a subroutine, and the escape sequence is a call statement. The qedx editor also allows the user to invoke a qedx macro from command level. To do this, the user merely places his macro in a segment that has the letters qedx as the last component of its name (e.g., alpha.qedx or alpha_1.beta.qedx). He can then invoke the macro by issuing the qedx command followed by the appropriate segment name. For example, to invoke the alpha_1.beta.qedx macro, the user types:

```
! qedx alpha_1.beta
```

A complete description of the qedx editor, including its buffer and macro capabilites, is given in the MPM Commands.

RESOURCE MEASURING

Through various commands, each Multics user can check his secondary storage quota usage; get information (including size, names, and access modes) about his segments, directories, multisegment files, and links; and print a month-to-date report of his resource consumption.

There are two commands directly related to secondary storage quotas. The getquota command allows the user to get information about the amount of secondary storage he may use. The movequota command moves all or part of a quota between two directories, one of which must be immediately inferior to the other.

Three commands print information about segments, directories, multisegment files, and links. The list command (briefly described in Section V) lists the names, access modes, time last used or modified, and the lengths of either all the entries in a directory or just selected entries. The count and total number of records occupied by either specific segments, directories, multisegment files, or links are printed by the listotals command. The listnames command prints a list of names of all the segments, directories, multisegment files, or links.

The user can issue the resource_usage command to print a month-to-date usage report for his own resources. He cannot issue this command to get information about any resource usage except his own. Through this command, the user can check his dollar charges according to shift and queue and also type of usage (interactive, absentee, or input/output daemon).

All of these commands are fully described in the MPM Commands.

RING STRUCTURE

As a further refinement of access control (see Section VIII), the system uses a special capability called the ring structure. This additional degree of protection, implemented by special hardware, is unique to Multics. Most users need not be concerned about the rings in which they will be working. Advanced users who require the use of ring protection for special data bases should see their project and system administrators.

Many system segments on Multics execute as part of a process, making calls and returns unknown to the user. These system segments must be protected from unauthorized modification by user segments. This protection is achieved by grouping segments into rings. Multics operation is controlled in such a way that procedure segments execute in a number of mutually exclusive subsets. These subsets may be considered concentric rings of privilege, representing different levels of memory access rights. The innermost or hardcore ring is made up of those segments essential to all users. This innermost ring, designated as ring 0, represents the highest level of privilege. The outermost ring, designated as ring 7, has the lowest level of privilege.

A procedure segment in an outer ring can call or pass data to, but cannot modify, a procedure segment in an inner ring. Normally, a procedure segment in an outer ring cannot access data in an inner ring. Within user-imposed limitations, an inner-ring segment can modify a segment in an outer ring. Every attempted access of one segment by another is checked for proper user access and for the ring of the referencing procedure segment to prevent invalid modification. The Multics ring-handling mechanism is enforced at the hardware level.

The ring structure capability permits the ready construction of protected data bases. In such a base, privileged users could be given the ability to read detailed information while nonprivileged users could receive only summarizations of the information. For example, a management information system could be developed that would allow management to designate which data and procedures could be accessible to different levels of personnel.

For more information about the ring structure, refer to the MPM Subsystem Writers' Guide under "Intraprocess Access Control (Rings)."

RUNNING OFF MANUSCRIPT FORMAT

The runoff command is used to type out text segments in manuscript form. Control arguments for the runoff command allow the user to completely regulate the processing of his text. For example, he can convert the output to be suitable to a particular kind of type ball or device, start or end the printing at a particular page, have source line numbers printed in the left margin, have the system wait for a carriage return before beginning and after each page of output, or direct the output to a special segment so he can print the material on a high-speed printer using the dprint command.

Runoff input segments must contain runoff control lines as well as text lines. There are over 50 types of control lines. They allow users to do such things as:

Specify up to 20 headers and 20 footers per page
Set the line length and page length
Have either roman or arabic page numbers
Skip a specified number of lines for an illustration
Center lines
Format equations
Control the size and number of margins
Set the spacing (single, double, or multiple)
Justify the margins
Translate a specified character to be printed as a different specified character in the output

This last feature is especially useful when the user wants a single blank between two character strings; the translate control line prevents runoff from splitting the strings between two lines or inserting padding spaces between the two strings. For complete information on all the control lines and the other capabilities of the runoff command, see the MPM Commands.

This document was prepared using the runoff command.

SETTING SEARCH CRITERIA

Whenever the user issues a command or references a program, the system must search through directories to find the specified command or program. The search is regulated; that is, certain specified search rules are followed by the system.

The default search rules (those automatically used by the system) may be changed and/or supplemented by the user. The set_search_rules command allows the user to change the default search rules, and the set_search_dirs command allows the user to insert search directories after the working directory in the default search rules. To check the current search rules, the user can invoke the print_search_rules command.

Adding another directory to be searched after the working directory is a convenient way for an entire project to share a group of special programs peculiar to the project's work. After a user on the project adds this special directory to his search rules, he can execute any of the programs in that directory as easily as he executes system commands. This addition to the search rules means that each user on the project saves himself the time and cost of either copying each one of the programs or linking to each one.

The user can determine whether a system command or a user-written command with the same name is to be used by setting his search rules (refer to Section IV).

All three of the commands governing the search rules are documented in the MPM Commands.

walk_subtree COMMAND

The walk_subtree command is used to execute a specified command line in a specified directory and in all directories inferior to the specified directory. Through various options, the user can state the first and last levels at which the command line should be executed, or he can have the command line executed in the lowest level directory first.

The walk_subtree command is an especially convenient way to list certain segments in a group of directories (for example, to list all of a user's runoff segments in all of his directories). Refer to the MPM Commands for a complete description of this command.

APPENDIX A

GLOSSARY

access attributes

Access attributes identify the kinds of access which may be set for a segment or directory. The access attributes for segments are read (r), write (w), execute (e), and null (n). Those for directories are status (s), modify (m), append (a), and null (n). Access attributes are also known as access modes. See Section VIII for more information.

ACL

An access control list (ACL) describes the access attributes associated with a particular segment. The ACL is a list of user identifications and respective access attributes. It is kept in the directory that catalogs the segment.

"carriage return"

A "carriage return" means that the typing mechanism moves to the first column of the next line. On Multics, this action is the result of the ASCII line-feed character. The terminal type determines which key(s) the user presses to perform the equivalent action (e.g., RETURN, LINE SPACE, or NL).

directory

A directory is a segment that contains information about other segments such as access attributes, number of records, names, and bit count.

directory (home)

The home directory (or the initial working directory) is the one under which the user logs in. Usually this directory is named:

```
>udd>projectid>personid
```

directory (working)

The working directory is the one under which the user is doing his work. Often the working directory is also the home directory. (This is always true at log-in time.) The user may redefine his working directory by use of the change_wdir command.

entryname

An entryname is the name by which a segment is cataloged in a directory. The entryname is not restricted to one component; it may contain two or more components, separated by periods.

link

A link is a name in a directory that points to a segment. A link enables a user to access a segment without using the normal search rules; i.e., given proper access permission, he may specify the segment by entryname (as though it were cataloged in the working directory) without actually having to make a copy of the segment. This is one of the ways in which Multics facilitates sharing.

page (also known as record)

A page is a unit of storage in Multics. A page contains 1024 36-bit words (4096 characters).

pathname

A pathname is the concatenation of a segment's entryname with all or some superior directories leading back to the storage system root.

pathname (absolute)

An absolute pathname is a concatenation of a segment's entryname with all superior directories leading back to the storage system root.

pathname (relative)

A relative pathname is a pathname that uniquely names a segment relative to the working directory; it may be the entryname portion of an absolute pathname (i.e., simply the entryname).

personid

A personid is an identification code under which a particular user is registered on the system. It is usually some form of the user's name and contains both uppercase and lowercase characters. It may not contain blank characters.

projectid

A projectid is an identification code under which a particular project is registered on the system.

quit signal

A quit signal is the means by which users may interrupt Multics from processing a program or command lines. The quit signal is invoked by pressing the ATTN, INTERRUPT, BRK, or QUIT key on the terminal; Multics responds with a ready message.

segment

A segment is the basic unit of information within the Multics storage system. Each segment has access attributes and a name and may contain data, programs, or be null.

APPENDIX B

STORAGE SYSTEM AND COMMAND CONVENTIONS

A number of conventions have been established for command names, command lines, arguments, and segment names. These conventions apply to most commands. However, some commands do not accept some conventions. Deviations are noted in the individual command descriptions in the MPM Commands.

COMMAND NAME CONVENTIONS

Command names never contain blanks. Commands that incorporate two or more words use an underscore (_) to separate words. Commands never have trailing underscores. (However, the majority of Multics subroutines do.) Most commands have an abbreviated name; the user may invoke the command by typing either the abbreviation or the full name. (This abbreviated name is independent of the abbreviation facility described in Section XI.)

COMMAND LINE CONVENTIONS

A command line consists of at least one command name and is terminated by producing an ASCII line-feed character. (Depending on the terminal, the proper key to use could be LINE SPACE, RETURN, or NL for new line.) The ASCII line-feed character is a signal to Multics to begin action on the typed command. Two or more commands (with or without arguments) separated by semicolons may be typed on a single line.

ARGUMENT CONVENTIONS

Commands do not always require arguments. When they do, arguments are separated from the command name (and from each other) by one or more spaces. An argument may contain blanks if it is enclosed in quotes (e.g., "Tom Smith").

If a command requires a specific number of arguments, failure to provide the proper number may result in incomplete or incorrect action. In such cases, an error message is printed followed by a ready message. For many commands, the order in which arguments are typed is significant.

Control arguments start with a hyphen (-) in order to differentiate between other arguments and to avoid ambiguity. A control argument specifies some modification to the type of action performed. In most cases the order of these arguments is unimportant.

PATHNAMES

Use of the Absolute Pathname

Most commands require an argument in the form of a pathname to specify the segment on which the command will act. The name that uniquely identifies a segment among all other segments in Multics is called an absolute pathname. The absolute pathname is used as an argument in commands when a relative pathname is awkward to use; e.g., the specified segment is not located "near" the user's working directory.

Use of the Relative Pathname

Relative pathnames are often used instead of absolute pathnames because they are shorter and thus more convenient to type. They are relative to the user's working directory rather than to the storage system root. Relative pathnames do not begin with the greater-than symbol, although some kinds of relative pathnames can contain the greater-than symbol.

Often it is easier to redefine the working directory than to type absolute pathnames as arguments. Most users redefine their working directories (using the change_wdir command) in order to be able to use relative pathnames. They need to be sure that they have access to segments in these directories, however.

ENTRYNAME

The simplest form of relative pathname is called an entryname. For example,

```
a
```

is a relative pathname for >udd>ProjA>TSmith>a if the working directory is >udd>ProjA>TSmith.

Each segment and directory may have more than one entryname. Usually, the user assigns a short entryname for typing convenience. For example, the entryname shown above, a, could be the short name for an entryname like algorithm_alpha_test.pl1.

LONGER RELATIVE PATHNAMES

A longer relative pathname might be

```
alpha>beta
```

where alpha is cataloged in the user's working directory, and beta is cataloged in directory alpha.

NAMING CONVENTIONS FOR MULTIPLE COMPONENT ENTRYNAMES

An entryname, the name by which a segment is cataloged in a directory, may not contain greater-than or less-than symbols or other special symbols such as asterisk or equals characters. Also, entrynames containing blanks should be avoided since the command language uses blanks to delimit command names and arguments. Many entrynames have several components, separated by periods. By convention, entrynames for source programs in Multics have as the last component the name of the language in which they are written.

SPECIAL SYMBOLS

The Less-Than (<) Symbol

Less-than symbols may take the place of directory names in a relative pathname. Each less-than symbol, like each greater-than symbol, denotes a hierarchy level; however, each less-than symbol indicates one level (one directory) back up the hierarchy, starting at the working directory and going up toward the root.

For example, in Figure B-1, if the working directory is TSmith, the absolute pathname for segment "a" would be:

```
>udd>ProjA>Rolf>a
```

A much briefer name, using the less-than symbol is:

```
<Rolf>a
```

where "<" represents the ProjA directory (one level back up the hierarchy from the TSmith directory), and the Rolf directory "catalogs" segment a.

The Star (*) Convention

The asterisk or star convention is used by standard Multics commands to reference groups of segments. If an asterisk is used for one component of an entryname, the command matches any name in that particular component position. For example,

```
*.pl1
```

matches any two-component entryname whose second component is pl1. And,

```
*.*.red
```

matches any three-component entryname whose third component is red.

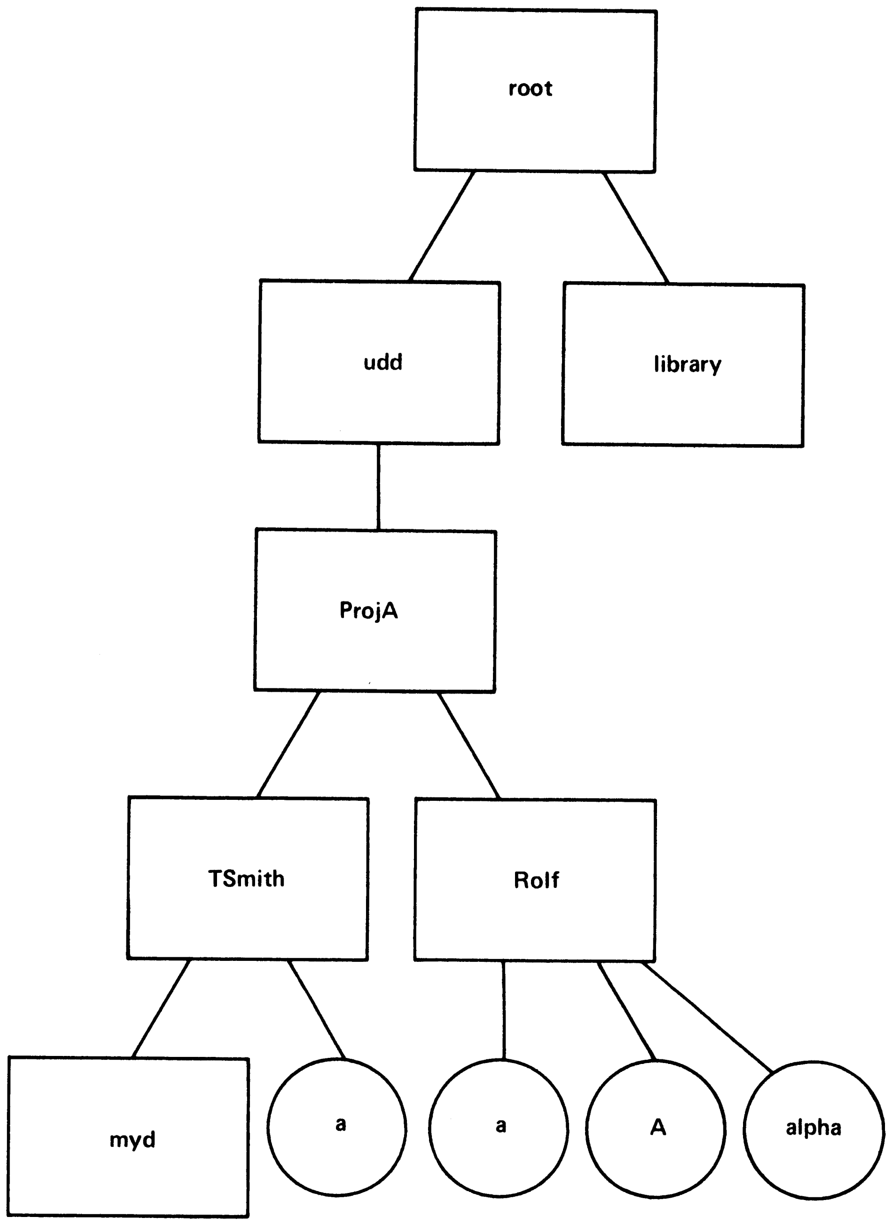

Figure B-1. Sample Hierarchy

A double asterisk may be used to match any number of components (including zero) on the right. For example, if the user wants a list of all the segments in his working directory with "blue" as the first component, he types:

```
! list blue.**
```

then, no matter how many other components exist in the entryname, the command considers them a "match":

```
blue
blue.red
blue.x.y
blue.x.y.z.n
```

The double asterisk may also be used to designate all entries in the specified directory. For example, to give read permission to Jane Doe for every segment in his current working directory, the user types:

```
! setacl ** r JDoe
```

Not all commands implement the star convention. The user should consult the MPM Commands under the proper command before using the star convention.

The Equals (=) Convention

The equals (=) symbol is used by standard Multics commands in the second of a pair of arguments to indicate equivalency with the same position component in the first argument. For example, if the user types:

```
! addname alpha.pl1 beta.=
```

the command interprets the second argument as beta.pl1. Thus, the segment may now be referenced as either alpha.pl1 or beta.pl1.

The double equals (==) sometimes appears as the second member of an argument pair. The double equals indicates equivalency with the same position component plus any other following components in the first argument. For example, if the user types:

```
! addname alpha.sort.pl1 beta.==
```

the command interprets the second argument as beta.sort.pl1. The segment may now be referenced as either alpha.sort.pl1 or beta.sort.pl1.

Not all commands implement the equals convention. The user should consult the MPM Commands under the proper command before using the equals convention.

APPENDIX C

REFERENCE TO COMMANDS BY FUNCTION

The Multics command repertoire is divided according to functions in the following pages. The 18 groups are:

Access to the System
Storage System, Creation and Editing of Segments
Storage System, Segment Manipulation
Storage System, Directory Manipulation
Storage System, Access Control
Storage System, Formatted Output Facilities
Storage System, Address Space Control
Language Translators, Compilers, Assemblers, and Interpreters
Object Segment Manipulation
Debugging and Performance Monitoring Facilities
Input/Output System Control
Command Typing and Control
Communication Among Users
Communication with the System
Accounting
Control of Absentee Computations
GCOS Environment
Miscellaneous Tools

Detailed descriptions of these commands, arranged alphabetically rather than functionally, are given in the MPM Commands or the MPM Subsystem Writers' Guide. In addition, many of the commands have online descriptions, which the user may obtain by invoking the help command as described in Section V.

ACCESS TO THE SYSTEM

enter	connects an anonymous user to the system (used at dialup only)
login	connects registered user to the system (used at dialup only)
logout	disconnects user from the system

STORAGE SYSTEM, CREATION AND EDITING OF SEGMENTS

adjust_bit_count	sets bit count of a segment to last nonzero character
convert_characters	performs character-by-character conversion on entire segment
create	creates an empty segment
edm	allows inexpensive, easy editing of ASCII segments

indent	indents a PL/I source segment to make it more readable
make_peruse_text	formats segment to use with peruse_text command
qedx	allows sophisticated editing, including macro capabilities (a minor interpreter)
sort_file	sorts ASCII segments alphabetically, line by line

STORAGE SYSTEM, SEGMENT MANIPULATION

archive	packs segments together to save physical storage
archive_sort	sorts the contents of an archive segment alphabetically, by component name
compare	compares segments word by word, reporting differences
compare_ascii	compares ASCII segments, reporting differences
compare_object	compares object segments, reporting differences
copy	copies a segment or multisegment file and its storage system attributes
create	creates an empty segment
file_output	directs terminal output to a segment
move	moves segment or multisegment file and its storage system attributes to another directory
reorder_archive	rearranges order of an archive segment according to the contents of a control segment
set_bit_count	sets the bit count of a segment to a specified value
truncate	truncates a segment to a specified length

STORAGE SYSTEM, DIRECTORY MANIPULATION

addname	adds a name to a segment, directory, link, or multisegment file
adjust_bit_count	sets bit count of a segment to last nonzero character
createdir	creates a directory
delete	deletes a segment or multisegment file and questions if it is protected
delete_dir	destroys a directory and its contents
deleteforce	deletes a segment or multisegment file without question
deletename	removes a name from a segment, directory, link, or multisegment file
fs_chname	renames a segment, directory, link, or multisegment file, bypassing naming conventions
link	creates a storage system link to another segment, directory, link, or multisegment file
list listnames listotals	prints directory contents
move	moves segment or multisegment file and its storage system attributes to another directory
names	moves or copies names from one storage system entry to another
rename	renames a segment, directory, link, or multisegment file

safety_sw_off	turns safety switch off for a segment, directory, or multisegment file
safety_sw_on	turns safety switch on for a segment, directory, or multisegment file
set_bit_count	sets the bit count of a segment to a specified value
set_max_length	sets the maximum length of a segment to a specified value
set_ring_brackets	sets the ring brackets of a segment to specified values
status	prints all the attributes of an entry in a directory
truncate	truncates a segment to a specified length
unlink	removes a storage system link

STORAGE SYSTEM, ACCESS CONTROL

delete_iacl_dir	removes an initial ACL for new directories
delete_iacl_seg	removes an initial ACL for new segments
deleteacl	removes an ACL entry
list_iacl_dir	prints an inital ACL for new directories
list_iacl_seg	prints an initial ACL for new segments
listacl	prints an ACL entry
set_iacl_dir	adds (or changes) an initial ACL for new directories
set_iacl_seg	adds (or changes) an initial ACL for new segments
setacl	adds (or changes) an ACL entry

STORAGE SYSTEM, FORMATTED OUTPUT FACILITIES

dprint	queues a segment or multisegment file for printing on the high-speed printer
dpunch	queues a segment or multisegment file for card punching
dump_segment	prints segment contents in octal, ASCII, or BCD
mail	prints or sends mail
memo	allows users to set reminders for later printout
print	prints an ASCII segment
print_motd	prints the system message of the day
runoff	formats a text segment according to internal control words
runoff_abs	invokes the runoff command in an absentee job

STORAGE SYSTEM, ADDRESS SPACE CONTROL

change_default_wdir	sets the default working directory
change_wdir	changes the working directory
initiate	adds a segment to the address space of a process
list_ref_names	prints all names by which a segment is known to a process
print_search_rules	prints names of directories searched for segments referenced dynamically

print_default_wdir	prints name of default working directory
print_wdir	prints name of current working directory
set_search_dirs } set_search_rules	allows users to modify search rules
terminate	removes a segment from process address space
where	prints absolute pathname of a segment

LANGUAGE TRANSLATORS, COMPILERS, ASSEMBLERS, AND INTERPRETERS

alm	translates assembly language programs
alm_abs	invokes the ALM assembler in an absentee job
apl	invokes the APL interpreter
basic	compiles BASIC programs
fortran	compiles FORTRAN programs
fortran_abs	invokes the FORTRAN compiler in an absentee job
pl1	compiles PL/I programs
pl1_abs	invokes the PL/I compiler in an absentee job
qedx	allows sophisticated editing, including macro capabilities (a minor interpreter)

OBJECT SEGMENT MANIPULATION

bind	packs two or more object segments into a single segment
compare_object	compares object segments, reporting differences
display_component_name	prints name and offset of a bound component
print_bind_map	prints information about a bound segment
print_link_info	prints list of entries and outbound links of an object sgment

DEBUGGING AND PERFORMANCE MONITORING FACILITIES

change_error_mode	adjusts length and contents of status messages
debug	permits symbolic source language debugging
dump_segment	prints segment contents in octal, ASCII, or BCD
error_table_compiler	compiles a table of status codes and associated messages
how_many_users	prints the number of logged-in users
page_trace	prints list of pages recently demanded
print_linkage_usage	prints map of all current linkage
profile	prints information about execution of individual statements within program
progress	prints information about the progress of a command as it is being executed
ready	prints the ready message: a summary of CPU time, paging activity, and memory usage
ready_off	suppresses the printing of the ready message
ready_on	restores the printing of the ready message
reprint_error	repints an earlier status message
trace	traces subroutine calls
trace_stack	prints stack history
where	prints absolute pathname of a segment

INPUT/OUTPUT SYSTEM CONTROL

cancel_daemon_request	cancels a previously submitted daemon request
close_file	closes open PL/I and FORTRAN files
console_output	restores terminal output to the terminal
dprint	queues a segment or multisegment file for printing on the high-speed line printer
dpunch	queues a segment or multisegment file for card punching
file_output	directs terminal output to a segment
ioc	allows direct calls to input/output system entries
list_daemon_requests	prints list of daemon requests currently queued
print_attach_table	prints list of current input/output system stream attachments

COMMAND TYPING AND CONTROL

abbrev	allows user-specified abbreviations for command lines or parts of command lines
answer	answers questions normally asked of the user
do	expands a command line with argument substitution
exec_com	allows a segment to be treated as a list of executable commands
get_com_line	prints the maximum length of the command line
new_proc	creates a new process with a fresh address space
program_interrupt	signals a condition following a quit or an unexpected signal
progress	prints information about the progress of a command as it is being executed
release	discards process history retained by a quit or an unexpected signal interruption
set_com_line	sets the maximum length of the command line
start	reenters process at point of a quit or an unexpected signal interruption
walk_subtree	executes a command in all directories below a specified directory

COMMUNICATION AMONG USERS

accept_messages	initializes the process to accept messages immediately
defer_messages	inhibits the normal printing of received messages
immediate_messages	restores immediate printing of messages
long_message_format	causes messages to be printed in verbose format
mail	prints or sends mail
print_messages	prints any pending messages
short_message_format	causes messages to be printed in brief format
send_message send_message_acknowledge send_message_silent	sends message to specified user
unlock_messages	unlocks a locked message segment

COMMUNICATION WITH THE SYSTEM

check_info_segs	checks information (and other) segments for changes
help	prints special information segments
peruse_text	prints special information segments in outline format
print_motd	prints the system message of the day
who	prints list of users and absentee jobs currently logged in

ACCOUNTING

getquota	prints secondary storage quota and usage
movequota	moves secondary storage quota to another directory
resource_usage	prints resource consumption for the month

CONTROL OF ABSENTEE COMPUTATIONS

alm_abs	invokes the ALM assembler in an absentee job
cancel_abs_request	cancels a previously submitted absentee job request
enter_abs_request	adds a request to the absentee job queue
fortran_abs	invokes the FORTRAN compiler in an absentee job
list_abs_requests	prints list of absentee job requests currently queued
pl1_abs	invokes the PL/I compiler in an absentee job
runoff_abs	invokes the runoff command in an absentee job
who	prints list of users and absentee jobs currently logged in

GCOS ENVIRONMENT

gcos	invokes GCOS environment simulator to run one GCOS job
gcos_sysprint	converts GCOS BCD sysout print file to ASCII file suitable for use with the dprint command
gcos_syspunch	converts GCOS BCD sysout punch file to file suitable for use with the dpunch command
gcos_utility	copies card image files, translating from GCOS format to ASCII or vice-versa

MISCELLANEOUS TOOLS

archive	packs segments together to save physical storage
archive_sort	sorts the contents of an archive segment alphabetically by component name
calc	performs specified calculations
code	enciphers segment, given a coding key
decode	deciphers segment, given proper coding key
reorder_archive	rearranges order of an archive segment according to the contents of a control segment

INDEX

Honeywell

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

16692, 3C1076, Printed in U.S.A. AL40, Rev. 0