HONEYWELL

NEW USERS' INTRODUCTION TO MULTICS — PART I

SOFTWARE

SERIES 60 (LEVEL 68)

NEW USERS' INTRODUCTION TO MULTICS — PART I

SUBJECT

Basic Introduction to Multics

SPECIAL INSTRUCTIONS

This manual is part of a two-volume set entitled *New Users' Introduction to Multics* (Order Nos. CH24 and CH25). The introductory set, along with one of the Multics text editor user guides, are prerequisites to all further Multics manuals. The text editor user guides are:

qedx Text Editor Users' Guide	Order No. CG40
Emacs Text Editor Users' Guide	Order No. CH27

SOFTWARE SUPPORTED

Multics Software Release 8.0

ORDER NUMBER

CH24-00

November 1979

Honeywell

PREFACE

The purpose of this manual is to help you become familiar with the Multics system. This manual provides you with a basic introduction to Multics, a workbook that guides you through your first sessions at a terminal. The topics covered here are fundamental Multics concepts that are immediately useful to the new user. Many examples are included, illustrating both correct and incorrect ways of sending instructions to Multics. Many of the concepts presented here are covered more thoroughly in Part II.

Section 1 of this manual introduces the Multics system.

Section 2 covers how to enter and leave Multics. Multics conventions are presented in Section 3, as you learn to use several basic commands (instructions). Here you are also introduced to communication with other users via Multics.

In Section 4 you learn to enter text or other information, in units referred to as segments, for storage and processing on Multics. Segments are discussed more fully in Section 5, which also includes commands allowing you to manipulate your own segments.

The organization of all users' segments is explained in Section 6. This section also suggests ways of organizing your own segments, and describes commands for using other people's segments. Commands concerning access control (permission to share segments with other users) are presented in Section 7.

A glossary of the terms used in this manual can be found in Appendix A. Appendix B contains a list of the commands introduced in the manual, including the correct usage and a brief description for each.

File No.: 1L13

CH24-00

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 you have become familiar with the concepts covered in this introductory guide. The MPM consists of the following individual manuals:

Reference Guide	Order No. AG91
Commands and Active Functions	Order No. AG92
Subroutines	Order No. AG93
Subsystem Writers' Guide	Order No. AK92
Peripheral Input/Output	Order No. AX49
Communications Input/Output	Order No. CC92

Throughout this manual, references are made to both the MPM Commands and Active Functions manual and the New Users' Introduction to Multics Part II. For convenience, these references will be as follows:

MPM Commands
Part II

The Multics operating system is referred to in this manual as either "Multics" or "the system". The term "computer" refers to the hardware on which the operating system runs.

CONTENTS

SECTION 1

INTRODUCTION

THE MULTICS SYSTEM

A large-scale computer is a machine that can store very large amounts of information, and can process that information very quickly. A computer consists of hardware, all the physical devices and electronic circuitry, and software, all the programs and other machine instructions that control the activities of the computer. The software is said to "run" or "execute" on the hardware.

In order for people to use a computer, there must be a set of programs within it that can interpret users' instructions, control the hardware, and otherwise supervise the basic operation of the computer. This portion of the software is called the operating system.

Your operating system is called Multics. It is a general-purpose system developed to serve large and diverse user communities. Because Multics is a timesharing system, the work of many users is processed almost simultaneously. The normal mode of operation on Multics is interactive, or "conversational": each instruction that you type goes directly to the computer and is acted on immediately; if Multics needs any further information from you in order to follow your instruction, you are asked for it. In addition, all of your work can be seen and used by other users through a system of access control: levels of permission controlled by you.

You will do your work on Multics from a terminal; in fact, this may be the only piece of hardware you see. A terminal looks very much like an electric typewriter (or a television screen) with extra keys. It is connected to the computer, either directly or through normal telephone lines. Although there are many different kinds of terminals, each with slightly different keyboards and instructions for use, all terminals can be classed as either printing terminals (also called "hard-copy" terminals because they print onto paper) or video terminals (with a screen and cathode ray tube, CRT, rather than paper).

MANUAL CONVENTIONS

A few conventions and special symbols should be introduced before you begin to explore the Multics environment.

Technical or other unfamiliar terms are underlined when used the first time, and are included in the glossary (Appendix A).

Quotation marks are used to indicate the exact spelling of a word, or the way a word should appear on a line typed by a user. For example, part of a person's identification on Multics is called a Person_id, and that term is unquoted, but a specific example of it, say "PSissle", is quoted. You do not type these quotation marks.

Another convention within examples is the use of an exclamation point to indicate lines you type. The exclamation point does NOT appear on your terminal -- you do not type it, and Multics does not type it to prompt you. Exclamation points appear ONLY in examples, and ONLY to show which lines you type.

Line numbers and text within angle brackets (<...>) are used within examples for explanatory purposes ONLY. They are not actually typed by Multics, and they should not be typed by you.

MULTICS CONVENTIONS

Several characters have special meanings on Multics.

Two special characters are deletion characters that you can use to correct typing mistakes. To delete single characters, use the "#" character (called the erase character) directly after the characters you want deleted, one "#" for each character you want to delete. Thus this line:

```
! My name is Pm#am.
```

is seen by Multics as:

```
My name is Pam.
```

The erase character, when used to erase a blank space, will erase ALL the blank spaces between characters. For example, this line:

```
! My namei     s### is Pam.
```

is seen as:

```
My name is Pam.
```

because the first # erases the "s", the second erases the next row of 4 blanks, and the third erases the "i".

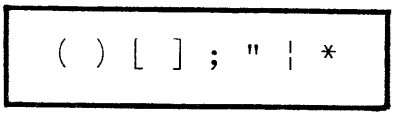
The "@" character (the kill character) deletes all characters to the left of it on the line; for example:

```
! Myn aemi s@My name is Pam.
```

is seen as:

```
My name is Pam.
```

You should avoid some characters unless you are specifically instructed that you may use them. These are:



You will learn about the correct use of these characters in Part II.

On Multics, names are not allowed to contain blank spaces. If a name consists of more than one word, the "_" (underscore) character is used to separate the words. For example, the send_message command name has an underscore connecting the two words. This convention is treated fully in Section 3.

Other characters have special meanings only in some contexts. Such characters are noted where necessary.

SECTION 2

ENTERING AND LEAVING MULTICS

Entering the Multics system is called logging in, and leaving the system is called logging out.

USER IDENTIFICATION ON MULTICS

To successfully log in on Multics, you must be registered as a Multics user at your site. When you get registered, you are assigned your own identifying name, called a User_id. Here is a sample User_id:

```
PSissle.Doc
```

There are two components, separated by a period, in this User_id: the Person_id (PSissle), which is a unique (to Multics) version of Pam Sissle's name, and the Project_id (Doc), an abbreviation of the name of the group she works with, which is included for administrative and accounting purposes. (A third component, the tag, is discussed in Section 7.)

Your User_id identifies you as a registered user, with authorization to use the resources of the system: computer time for interacting with the system and space (memory) to store the work you've done. Although your User_id is unique, it is public and can be known by everyone.

You also receive a preliminary password when you register. One of the first things you learn to do on the system is to change the assigned password to one you make up, so that it will be completely private. Your password ensures that only you can log in with your User_id because you are the only person who knows what your password is. The password system is the most important key to the security that Multics offers.

CONNECTING THE TERMINAL

Now you are a registered user, sitting at a terminal. After turning the terminal on (there is an on/off switch on the body of the terminal), you must make an electronic connection between it and the computer. Before you make this connection, the terminal is just another electric typewriter, with a few extra characters on the keyboard. (If you have a hardwired terminal -- one that is connected directly and permanently to the computer -- you are ready to turn to the next part of this section and log in.) This procedure is simple, but it differs slightly at each site. Your site probably has detailed instructions available. Here is a general description of terminal connection.

Terminal connection is made through normal telephone lines using a modem. The modem sends electronic signals to and from the terminal in the form of high-pitched tones. It may be built into the terminal, or it may be a separate unit connected by a cable to the terminal. If it is built in, you see only a cradle that accepts a telephone receiver, and there is a regular telephone nearby. Otherwise, you use a dataset, which looks very much like a regular telephone with several pushbuttons along it.

Pick up the telephone receiver, listen for a dial tone, and dial the phone number of your system (this number should be included with the instructions for your particular site, and may also be posted near the terminal). You will hear the phone ringing, and then a steady shrill tone when the connection is made. At this point, if you have a built-in modem, place the receiver firmly into the cradle, with the cord positioned as indicated on the unit. If your modem is a dataset, push the indicated button (labeled "DATA" or "HOLD") and then set the receiver back into its place on the modem. Usually, either the modem or the terminal turns on a small light when the connection is made. When this light goes off, it means you are disconnected.

Terminal connection sounds complicated, but it will become automatic for you very soon. If you have problems, make sure everything is plugged in and turned on, and ask another user for assistance.

Logging In

When you have successfully connected your terminal, Multics prints a banner, or greeting message. This is the signal for you to log in and identify yourself. Here is a sample login sequence for Pam Sissle, opening with a typical site banner. The lines are numbered for this example ONLY; the example is followed by an explanation of each line.

```
1.    Multics MR8.0: Honeywell LISD Phoenix, System M
      Load = 42 out of 125.0 units: users = 39. 05/02/80 ...
2.  ! login PSissle Doc
3.    Password:
4.  ! pwsWWWWWWWWW
5.    You are protected from preemption until 1007.8
6.    PSissle Doc logged in 05/02/80 0907.5 mst Fri from ...
7.    Last login 05/01/80 1326.4 mst Wed from ...
8.    r 10:07 1.486 34
```

NOTE: Remember that you do NOT type exclamation points or line numbers to begin your command lines. Exclamation points are provided in this manual only as an aid to readers, to indicate lines that are typed by a user. Line numbers are used only for purposes of explanation.

1. The typical Multics banner is a two-line message giving the current version of Multics software, the name and location of your site, the number of people logged in, the total number of users the system can accept at one time, and the date and time.

2. The word "login" is the command (a command is a specific instruction to the computer) that identifies you and gives you access to Multics. Actions that you want performed by Multics are conveyed by commands.

   Follow the login command with a space and your own Person_id and Project_id. Then press the RETURN key (this key may be labeled "RET" or "CR" on your terminal). All lines sent to Multics must be terminated by typing the RETURN key or else Multics will not act on them. This is a complete command line.

3. The login command is an interactive command: it responds to your Person_id and Project_id by requesting your password.

4. Both you and Multics use this line. First Multics prepares to conceal your password, so that nobody can read it. Depending on the kind of terminal you have, the printing of your password is either suppressed entirely or hidden in a string of cover-up characters. Then you type your password, and the RETURN key.

   In this example, Pam Sissle typed in her preliminary password of "pws", her initials (all lowercase). Pam's password is left visible in these examples to illustrate the correct response to the login password request.

5. This message tells you how much time you are guaranteed on the system during this login session; after the stated time (here it is 10:07 am), you may occasionally be logged out (with advance warning), but usually you have as much time as you want. This line also serves as acknowledgment of a successful login.

6. Your User_id and the date, time, day and location of your current login session are listed.

7. The date, time, day and location of your last login session are listed. You can use this information to detect unauthorized use of your User_id and Person_id and Project_id. Multics will also inform you here of any unsuccessful attempts to log in using your Person_id and Project_id (including your own attempts, if you misspell your password).

8. The last line is a ready message. Included in this is the current time of day (on a 24-hour clock), and other information that reflects your use of system resources. The ready message appears when you are at command level: whenever Multics is ready to receive another command from you.

Now you are ready to begin working on Multics.

LOGIN PROBLEMS

It is possible that the first time you type the login command line, you will not receive the prompting message "Password:". In fact, almost nothing will happen: your carriage (or cursor, the white square or blinking underscore on video terminals) moves over to the beginning of the login line without moving down to the next line. In this case, also press the LF (or LINEFEED) key. After you receive the prompting message and type your password, press both the RETURN and the LF keys again. When you receive the ready message from Multics, type:

```
set_tty -modes lfecho,crecho
```

and both the RETURN key and the LF key. From now until you log out, you can type just the RETURN key to send your command lines to Multics.

You will learn more about the set_tty command in Part II, but for now, if you do not want to type this command line every time you log in, ask your project administrator to make your RETURN key include a line feed permanently. (You can also do this yourself after reading the "exec_com" section in Part II.)

If you make a mistake while logging in, and you have already typed RETURN without using the erase or kill characters, Multics tells you by printing an error message, and asks you to try again. Several error messages are shown in this section. Multics sends you an error message whenever you send it a line that it cannot process, because of incorrect format or inadequate information. No harm has occurred when you receive an error message; check your command line and try it again.

Here is an example of a typing mistake (Psissle rather than PSissle) on the command line:

```
! login Psissle Doc
  Password:
! pwsWWWWWWWWW
  The user name you supplied is not registered.
  Please try again or type "help" for instructions.
! login PSissle Doc
```

Although the error was on the first line, Multics checked for a matching password before sending an error message. When you try again, start with the login command line.

Typing mistakes are the most common errors. It is important to type words exactly as they are indicated, with attention to both uppercase and lowercase letters (Multics distinguishes between uppercase and lowercase).

Here is another example, in which Pam forgot the space after the command:

```
! loginPSissle Doc
  Incorrect login word "loginPSissle".
  Please try again or type "help" for instructions.
! login PSissle Doc
```

Another easy mistake to make is to type your password incorrectly. (It is also hard to correct a mistyped password using the erase character, because you cannot see what you typed.) Multics responds this way:

```
!  login PSissle Doc
   Password:
!  owsWWWWWWWWW
   Incorrect password supplied.
   Please try again or type "help" for instructions.
!  login PSissle Doc
```

After you mistype your password, you must type the entire login sequence again, beginning with the login command.

Each site administrator sets a limit to the amount of time you have and the number of attempts you can make to log in during one session; you usually have about 6 minutes or 6 tries. When you have exceeded this limit, Multics tells you to hang up the telephone, because it is automatically breaking your terminal connection:

```
!  login PSissle Doc
   Password:
!  owsWWWWWWWWW
   Incorrect password supplied.
   hangup
```

If you typed everything correctly but are still denied entrance to Multics, you may not be registered yet. Check with your project administrator if you think this is the case. There may be other reasons for denying you access, such as a system shutdown, for which you receive an explanatory message from Multics.

Logging Out

When you finish your work on Multics, wait for a ready message and type the logout command to break your terminal connection. Multics responds by printing your User_id, the date and time that you log out, and your total system resource usage. It then reminds you to hang up the telephone:

```
   r 11:25 0.072 68
!  logout
   PSissle Doc logged out 07/26/80 1125 mst Fri
   CPU usage 13 sec, memory usage 5.6 units, cost $2.58.
   hangup
```

Do not hang up the telephone before Multics has logged out; wait for the word "hangup".

You should always log out, hang up the telephone, and turn off the terminal before leaving, to avoid wasting computer time and to allow others to use the terminal. If you do not log out, another person can issue commands that Multics will interpret as being from you. This activity is charged to you, and may also result in damage to your work.

Login/Logout Variations: Control Arguments

Most commands are flexible, letting you vary the way the commands work. The command name specifies an action to be taken; to direct the action of the command, you supplement it with information by typing words called arguments after the command name. The different kinds of arguments are discussed fully in Section 3, but one kind is of special interest to you at this point: the control argument.

The control argument is distinguished from other kinds of arguments by the hyphen which always precedes it. It is used as an optional argument -- one that you include on the command line if you want the variation it stands for. Both the login and the logout commands take control arguments. You are introduced to a few of the most useful ones here.

The most important control argument for the new user, used with the login command, is -change_password. As discussed above, it is very important that you be the only person who knows your password. Here is an example of how to change a password:

```
! login PSissle Doc -change_password
  Password:
! pwsWWWWWWWWW
  New Password:
! newpassWWWWW
  New Password Again:
! newpassWWWWW
  Password changed.
  You are protected from preemption ...
  PSissle Doc logged in 07/26/80 ...
  Last login 07/26/80 ...
  r 11:27 2.019 889
```

Your password must be 8 characters or fewer (with no blanks). You may change your password as often as you like, but do not choose a word that people are likely to guess, such as your initials or the name of a pet. Try not to forget the one you use, because there is no record of it available to any person. If you do forget it, you must contact your project administrator and request a new one.

Another useful control argument for the login command is -brief. If you use the terminal several times a day, you may not want to see the introductory information that follows receipt of your password (lines 5 through 7 in the first example). Here is an example of logging in using the -brief control argument:

```
! login PSissle Doc -brief
  Password:
! newpassWWWWW
  r 11:29 1.798 699
```

The logout command also accepts a -brief control argument:

```
! logout -brief
  hangup
```

When used with the logout command, -brief suppresses the logout banner.

The -hold control argument to the logout command aids the person who uses your terminal after you. It retains the terminal connection after you have logged out, and prints the Multics banner in readiness for another person to log in:

```
! logout -hold
  PSissle Doc logged out ...
  CPU usage ...

  Multics MR8.0: ...
  Load = 23 out of 60 ...
```

You can include several control arguments with one command, as long as they are acceptable to the command and are typed after the command but on the same line. For example:

```
! login PSissle Doc -change_password -brief
  Password:
! pwsWWWWWWWW
  New Password:
! newpassWWWWW
  New Password Again:
! newpassWWWWW
  Password changed.
  r 11:37 2.381 1121
```

When you combine the two control arguments to the logout command, the results may surprise you. Here is the complete interaction, from ready message to new login -- Pam is trying this out by logging in again:

```
  r 11:39 0.048 32
! logout -brief -hold
  <blank line from Multics here>
! login PSissle Doc
```

The -brief control argument carries over into the new login session -- no banner is printed! You must still use the "login -brief" command line if you want the shortened version of your new login session, though. The only indication you have that Multics is ready for a new user is that Multics sends a blank line after it logs you out.

As you learn about more commands, read through the command descriptions (Appendix B) to find out about the control arguments available to you with each command.

SECTION 3

BASIC MULTICS COMMANDS

COMMAND CONVENTIONS

Nearly all work that you want to accomplish on Multics is conveyed by commands. There are over 400 commands, many of which you may never use, some that you will use often. Most commands share several established rules of usage called conventions. (Some commands do not conform to some conventions -- exceptions are documented in the individual command descriptions, Appendix B.) Several basic commands are introduced to illustrate the concepts and conventions presented in this section.

Command Names

Multics differentiates between uppercase and lowercase letters. Command names are always typed in lowercase letters. For example, when you log in to the system you type "login". If Pam Sissle types "LOGIN" instead, Multics responds with an error message:

```
! LOGIN PSissle Doc
  Incorrect login word "LOGIN".
  Please try again or type "help" for instructions.
```

Command names never contain blanks. Another way to say this is that a command name is always one character string: one group of alphabetic, numeric, and some special characters (periods, hyphens and underscores) unbroken by blanks. When two or more words are incorporated in one character string, an underscore is used to simulate blanks between words, as with the "send_message" command or the "-change_password" control argument.

Many command names have shortened versions, called short names, that can be used interchangeably with the full names. The login command has the shortest short name, "l" (the logout command has no short name). Pam Sissle can log in this way:

```
! l PSissle Doc
```

Many of the most common control arguments have short names, too. Here is the full version and the shortened version of a login line from Section 2:

```
! login PSissle Doc -change_password -brief
! l PSissle Doc -cpw -bf
```

Short names are given in command descriptions (Appendix B), next to the full names of those commands and control arguments that have shortened versions. In this manual, short names also appear in parentheses after command names.

Command Lines

A command line consists of a command name, any desired or necessary arguments (separated from each other by blanks), and a newline. You must use blanks to separate the name of the command from its arguments, and to separate arguments from each other. Multics processes command lines in the order that they were typed.

Every command line ends with a newline. The newline functions as a simultaneous carriage return (returning you to the beginning of the same line) and linefeed (dropping you to a new line). These two actions are a signal to Multics that you have completed a command line and want it acted on. The newline is sent to Multics by typing the key marked "RETURN" (some terminals have a different key for this function -- check at your site for the correct key). The RETURN keys on most terminals act as newlines automatically; if your RETURN key does not (if you must accompany it with a linefeed to send your command line to Multics), see "Login Problems" in Section 2 or talk to your project administrator.

Stopping a Command

If you want to stop a command while it is acting, you can do so by issuing a QUIT signal. The who comand provides a good example. The who command prints a list of all the users currently logged in, preceded by a Multics banner like the one you see when you log in. When you type "who" to find out who is logged in, you may not want to see the entire list of current users. As soon as the system displays the names you are interested in, you may issue a QUIT signal by pressing the "QUIT" key only (this key may be labeled BREAK, ATTN, or INTERRUPT on your terminal). When Multics receives the QUIT signal (there may be a short delay), it interrupts the command and displays a QUIT message:

```
!  who

   Multics MR8.0, load 32.0/40.0; 32 users, 28 interactive,
     2 daemons.
   Absentee users 2/3
   IO.SysDaemon
   Dumper.Daemon
   Roach.SysMaint
   DAdam.SoftWork
   PSissle.Doc            <she presses QUIT here>
   QUIT
   r 13:14 0.099 59 level 2
```

At this point, you are at command level 2; Multics has created a new command level for you. To return to command level 1, you should type release -all:

```
      r 13:14 0.099 59, level 2
!     release -all
      r 13:15 0.062 41
```

When Multics prints a ready message with no level number, you are back at command level 1, which is where you started. For more information on the QUIT signal, see Part II.

ARGUMENT CONVENTIONS

Arguments are character strings included on the command line that provide any information necessary for a command to act as you want it to. The way arguments are typed in a command line is shown on a syntax line in the individual command descriptions (Appendix B). A generalized example of a syntax line is:

```
command argument1 argument2 argument3
```

for as many arguments as the command accepts. Each command has its own list of the arguments that it can accept; these lists of arguments are also included in command descriptions.

One command may accept several different kinds of arguments; each kind is discussed below. When several arguments can be accepted on one command line, the order in which they are typed is often significant. (Such requirements are indicated in the syntax lines.) If you type arguments in the wrong order, or if you type fewer arguments than necessary for the command to do what you want, Multics usually sends you an error message pointing out your mistake:

```
! whom
  Segment whom not found.
  r 13:18 0.068 98
```

If your mistaken command is still understandable to Multics, the command acts on whatever information you gave it. In this case, the results will probably not be what you intended.

The Standard Argument

One kind of argument is called simply an argument. It can be any item on which the command performs its action. Thus, such items as numerical values, the Person_id, and the Project_id function as arguments when they are called for in a command line. You have already seen two examples of this kind of argument in the login command line:

```
! login PSissle Doc
```

This syntax line in the "login" command description would be:

```
login Person_id Project_id
```

followed by a description of what words could be used as the arguments -- in this case the user's Person_id and Project_id.

These two arguments are required with the login command; if you don't include them you will receive a message reminding you to do so. Other arguments, those placed in braces on the syntax line, are optional. Here is an example of a command syntax line that includes an optional argument:

```
calendar {date}
```

The braces ({}) indicate that the enclosed argument is optional. If you type only "calendar" on the command line, the system prints out, at your terminal, a calendar of the current month. If you type the command name and a date of the form MM/DD/YY:

```
! calendar 03/01/72
```

you receive a calendar for the month you specified. (Note: you must be using a terminal that has 120 or more characters per line for the calendar to be printed correctly.)

Pathname Arguments

The most common kind of argument is the pathname argument. All information that you create and store on Multics is grouped into segments. For example, this section of this manual is stored in a segment named "section_3". You give your segments names, called pathnames, that tell the system where your segments are located. (Segments are discussed in more detail in Sections 4 and 5, and pathnames in Section 6.)

Many commands are designed to act on segments; for these commands, you provide the pathname of the appropriate segment on the command line. When you type the name of this segment as one of the arguments to a command, the system finds the segment by its pathname and gives it to the command to act on.

The syntax line for a command with a pathname argument is:

```
command path
```

or, if the pathname is optional:

```
command {path}
```

An example of a command that requires a pathname argument is the print command (short name "pr"), which prints a header and the text of Pam Sissle's segment "tadpole" at her terminal:

```
!  print tadpole

                tadpole 05/02/80 1322.5 mst Mon

    Selective listing:  Class Amphibia, Superorder Salienta,
    Order Anura, Suborder Procoela.
    <rest of text>
```

In the print command, the order of arguments on a command line is significant. The print command takes two optional arguments: line numbers that specify where to begin and end printing. You give these arguments if you do not want your whole segment printed (when using line number arguments, the header is not printed). If you want to use these arguments, you must type them after the pathname. For example:

```
!  print tadpole 23 25
   Family Atelopodidae
        Atelopus pachydermus
        A. carrikeri
   r 13:23 0.067 67
```

This restriction is conveyed by the order of the arguments in the syntax line:

```
print path {begin_line} {end_line}
```

Such restrictions are usually explained in the text of a command description.

Control Arguments

With all of the arguments described above, you replace the argument terms given in a syntax line -- "Person_id", "date" -- with the information you want the command to act on -- "PSissle", "10/17/80". Control arguments represent the special features of a command. With control arguments, you always type the same control argument (preceded by a hyphen, of course) to get that particular modification. Because control arguments cause a command to act differently than it normally would, they are nearly always optional. See "Variations" in Section 2 for several examples of control arguments.

Control arguments rarely need to be typed in a given order. However, control arguments do sometimes take arguments themselves, and there may be various kinds of restrictions to this kind of argument. One example of a command with a control argument that takes an argument was shown in Section 2, "Login Problems":

```
! set_tty -modes lfecho,crecho
```

where the -modes control argument takes the character string "lfecho,crecho" as an argument. Restrictions on the order of control arguments in a command line are not revealed by the syntax line of the command description. For example, here is the syntax line of the set_tty command (short name "stty"):

```
set_tty {-control_args}
```

Such restrictions are explained in the description of the relevant control argument.

Default Arguments

The last kind of argument is the default argument, an argument that the system assumes is present if you have typed nothing but a command name on the command line. Default arguments are assumptions about what the user wants from a command. The print command (pr) offers a clear example of a default argument. If your segment named "tadpole" has 39 lines in it, then typing:

```
! print tadpole
```

is the same as typing:

```
! print tadpole 1 39
```

(except that the header is suppressed when you specify line numbers).

Very few commands have no arguments at all -- many commands have default arguments.

COMMUNICATING WITH OTHER USERS

Multics offers several commands that allow you to communicate with other users. This facility -- known as the mail facility -- is very useful for such purposes as requesting permission to look at or use someone's segments, informing other users of upcoming events, or asking other users for assistance.

The mail facility commands are described below in two groups: message commands and mail commands. Message commands are intended primarily for interactive exchanges between two logged-in users. Mail commands provide the advantage of writing, sending and reading mail at your convenience, whether the other user is logged in or not. In order to receive either type of communication, you must have your own mailbox, a specially protected segment with the name Person_id.mbx (for example, Pam's mailbox is named PSissle.mbx). Your first use of the accept_messages command creates a permanent mailbox for you automatically.

RECEIVING MESSAGES

In order to receive upcoming messages from other users, you must type the accept_messages command (short name "am") each time you log in. In addition, to receive the messages that have been sent to your mailbox since you last logged out, include the -print control argument on the command line. The accept_messages command allows subsequent messages from other users to be printed at your terminal instantaneously. The User_id of the sender is included with each message, as well as the date and time:

```
! accept_messages -print
  From JCMulty.Doc 05/01/80 1107 mst Tues:
    Welcome to computers!
```

After your messages are printed at your terminal they are deleted from your mailbox.

Incoming messages are printed on your terminal regardless of what you may be working on. Although messages can interrupt you, they have no adverse effect on what you are doing. If you have only half a command line typed when a message arrives, simply read the message and then continue typing from where you were stopped -- the message is not sent to the system as part of your command line. For example, here Pam receives a message while she is typing the who command. Notice that until Pam presses the newline, her command line is not sent to the system.

```
! who          <she has not typed a newline yet>

  From JTKissle.SoftWork 05/02/80  0828 mst Wed: Hello!

! -brief       <newline now>
  IO.SysDaemon
  Backup.Daemon
  Roach.SysMaint
  Abelian.Groups
  JKLisa.SoftWork
```

If a message arrives while the system is printing out information, the message is printed and the system returns to printing as if nothing else had happened:

```
DAdam.SoftWork

From JTKissle.SoftWork 05/02/80  0829 mst Wed:
I just got registered today, too.

FBar.HardWork
JCMulty.Doc
PSissle.Doc
JTKissle.SoftWork
JMLie.Groups
Galois.Groups
```

There may be times when you don't want your work to be interrupted by messages. You can have your incoming messages collected and saved in your mailbox by typing the defer_messages command (dm). When you are ready to receive messages at your terminal, just type the accept_messages command, with the -print control argument, again.

You automatically stop accepting messages at your terminal when you log out. Remember that you should type this command:

```
! accept_messages -print
```

as soon as you log in each time. (You can have this done automatically after reading the "exec_com" section in Part II.)

SENDING MESSAGES

Interactive messages are sent with the send_message command (sm). There are two ways to send messages, as the syntax line implies:

```
send_message User_id {message}
```

For a short message, you can type the whole command line, including your message, on one line:

```
! send_message JTKissle.SoftWork Hello, there.
```

You may want to type a long message. When you type just the command, the user_id, and a newline, the system responds with the word "Input:". Now you are in input mode, the second form of the send_message command. As long as you remain in input mode, each line you type is sent immediately to the user you specified. To leave input mode and return to command level, type a period as the first and only character on a line, and then type a newline. A ready message will follow.

```
! send_message JTKissle.SoftWork
  Input:
! Are you reading the New Users' Intro?
! I'm just learning how to send messages - can I
! practice on you?
! .
  r 14:10 0.084 117
```

When you are in input mode, Multics expects normal text from you, rather than commands. It recognizes only the erase and kill characters as special symbols. You may therefore use any of the other special characters listed in Section 1, even though you should not use them when your message is included on the command line. In fact, you may even type command lines as messages, and they won't be acted on. Here is an example of both these features:

```
! send_message JTKissle.SoftWork
  Input:
! Say, Jakob, try typing this (it's really neat):
! sm JTKissle.SoftWork I know your password!
! .
  r 14:13 0.229 252
```

When you and another user both enter input mode, you can "chat" back and forth with each other:

```
! send_message JTKissle.SoftWork
  Input:
! Are you reading the New Users' Intro?
  From JTKissle.SoftWork:
  Yes - I see you also got to Section 3.
! What are you using Multics for?
  From JTKissle.SoftWork:
  I'm storing data from my research on infant behavior.
! Very interesting.  Well, back to Section 3.
! .
  r 14:19 0.378 327
```

Remember the period that takes you out of input mode! The period is easy to forget, and when you do, you send command lines to another user instead of to the computer.

The send_message command is an interactive command, the same way that Multics is an interactive system. You talk to Multics using commands, and Multics responds with requested information and ready messages. An interactive command talks to you by means of prompts, like "Input:" or "Password:", and you talk to it using requests, like the period that terminates input mode in the send_message command. After you type the command line and a newline for interactive commands, you are at request level: the system expects either appropriate information (like messages or a password) or a request from that command's list of accepted requests.

Mail Commands

The mail commands, print_mail (prm) and send_mail (sdm), are more complex interactive commands than the send_message command. Both commands have their own sets of requests for use within the command. Your first use of print_mail creates a permanent mailbox for you automatically, if you do not already have one.

Once you have a mailbox, you can receive mail as well as messages. Incoming mail always goes directly to your mailbox. If you are logged in and are accepting messages, you automatically get a notice each time new mail arrives:

```
From JCMulty.Doc 05/02/80 1213.7 mst Wed: You have mail.
```

You can read your mail at any time by typing the print_mail command (prm). The system tells you how many messages you have (within the mail facility, mail is referred to as messages), and prints the messages, with banners, one at a time. After each message is printed, the system sends you a prompting message and waits for you to type a request word in response. For example:

```
! print_mail
  You have 1 message.

  #1 (3 lines)  05/01/80 9:16  Mailed by: FBar.HardWork
  Date:  2 May 1980 1210 mst
  From:  FBar.HardWork
  Subject:  Meeting

  REMEMBER -- there will be a meeting of all
  people involved in the company carpool plan,
  on Thursday at 10am in Davis Adam's office.

  print_mail: Delete #1?  !  <request here>
```

Five requests are accepted in answer to the prompt:

yes
: deletes that message and prints the next one

no
: retains that message and prints the next one

reprint
: repeats that message and the prompt

quit
: retains that message and any messages not yet seen, and returns you to command level

abort
: retains ALL current messages (even those you have deleted) and returns you to command level

Keep in mind that you may save mail in your mailbox as long as you want. When you have answered the prompts for each message, you are returned to command level. A complete example follows:

```
! print_mail
  You have 3 messages

  #1 (7 lines) 05/02/80 10:02 Mailed by: JCMulty.Doc
  Date: 2 May 1980 10:02 mst
  From: JCMulty.Doc
  Subject: Greetings

  Welcome onto the Multics system; once you learn a few
  of the basics, I think you'll find it very useful.

  I'm your project administrator. If you have any
  questions or problems, please feel free to talk to me.

  Have fun!

  print_mail: Delete #1? ! yes

  #2 (2 lines) 05/02/80 10:42 Mailed by: JTKissle.SoftWork
  Date: 2 May 1980 10:42 mst
  From: JTKissle.SoftWork
  Subject: practice

  It's me again, practicing the send_mail command.
  That's all I can think of to say right now - bye.

  print_mail: Delete #2? ! quit
  r 10:54 0.394 489
```

Pam deleted her first message, read her second one, and returned to command level, so now she has two messages in her mailbox.

A useful control argument to the print_mail command is "-list", which prints a summary of your messages before printing the text of each one. Here is the summary of Pam's remaining messages:

```
! print_mail -list

  Msg# Lines   Date    Time   From                 Subject

    1   (2) 05/02/80 10:42  JTKissle.SoftWork   practice
    2   (1) 05/02/80 10:49  JTKissle.SoftWork   Junk Mail
```

To write and send mail, use the send_mail command (sdm). Because this is a powerful command, you may use it at any one of several levels of complexity. Two ways of using send_mail are shown here.

The easiest way to send mail is to type "send_mail" and the User_id of the person you want to write to. The command prompts you for a title and for the text of the mail; at this point you are at send_mail request level, in input mode. Conclude your mail in the same way as you terminate input mode when using the send_message command (sm): type a period as the first character on a line, and then type a newline. You now receive a message telling you that your mail has been sent, and you are returned to command level. This is how Jakob Kissle sent his first piece of mail to Pam Sissle:

```
!  send_mail PSissle.Doc
   Subject:  !  practice
   Message:
!  It's me again, practicing the send_mail command.
!  That's all I can think of to say right now - bye.
!  .
   Mail delivered to PSissle.Doc
   r 14:47 0.487 675
```

You may want to make changes to your mail before you send it. You can edit the message after you type it in, by typing "\f" (backslash-f) instead of a period when you end the message. Now you are in send_mail edit mode! It is very similar to the qedx edit mode, EXCEPT you do not need to use the "w" request -- that is done automatically. (For information about qedx edit mode requests, read Section 4, "Creating and Editing Text".) To leave edit mode, type "q". Rather than returning you to command level, the send_mail command stays at request level. It allows you several choices by prompting you for a request. The three most useful requests are:

print
: prints the mail you just typed

send
: sends the mail to the mailbox of the user specified on the command line

quit
: leaves send_mail request level and returns to command level.

Here is the way Jakob wrote his second piece of mail to Pam:

```
!  send_mail PSissle.Doc
   Subject:  !  Junk Mail
   Message:
!  I am doing more practicing.
!  \f
!  s/I am doing/This is/
!  p
   This is more practicing.
!  q
   send_mail:  !  send
   Mail delivered to PSissle.Doc
   send_mail:  !  quit
   r 15:08 ...
```

A variety of other requests are available for both the send_mail command (sdm) and the read_mail command. (The read_mail command is more powerful than print_mail for manipulating your incoming mail.) When you are comfortable with the capabilities introduced here, try out some of the options listed in the descriptions of these commands in the MPM Commands.

HELP COMMANDS

The help Command

The help command is an extremely useful interactive command that enables you to obtain information at your terminal about any given Multics command. The syntax line of the help command is:

help {command_name} {-control_args}

Typing the help command with the name of another command (and no control arguments) causes Multics to begin printing the info segment about the command you requested: this is an "information segment" consisting of blocks of information about a given command. The first block informs you of how long that first block is, how long the entire info segment is, the full name and short name of the command, the date on which the command was last modified, and the syntax line and function of the command. After each block, Multics prints the title of the next block and asks you if you want more help:

```
! help print
>doc>info>pr.info   (6 lines follow;  25 in info)
01/27/76  print, pr

Syntax:  pr path {begin_line} {end_line}

Function: prints an ASCII segment.

Arguments required (13 lines).  More help?  !  <request>
```

Some of the requests you may use are:

yes
: print the next block

skip
: skips the next block

rest
: prints the rest of the info segment, and returns you to command level

quit
: stops printing information, and returns you to command level

Two of the control arguments to the help command, added to the command line after you list the desired info segments, are also particularly useful:

-brief
: prints a summary of the command information, including the syntax line and all arguments and control arguments

-all
: prints the entire info segment without prompting you

Try the help command with the name of one of the other commands you have worked with. You can use it to remind yourself of the correct syntax line of a command, or to learn about arguments and control arguments not discussed in this manual.

The list_help Command

There are more than 500 info segments available to you at your terminal. The list_help command gives you a list of info segments that pertain to a given topic. The syntax line for this command is:

```
list_help {topic} {-control_arg}
```

It lists all the info segments whose names contain the given topic. For example, this command line:

```
! list_help help
```

produces this list:

```
help.changes
help
help_
info_seg_format.gi
```

Typing the list_help command with just the -all control argument:

```
! list_help -all
```

gives you a (very long) list of all the info segments.

SECTION 4

CREATING AND EDITING TEXT

TEXT EDITING COMMANDS

After logging in and becoming familiar with the basic Multics environment, you will probably want to create and edit information on Multics.

Multics has several text editors, powerful interactive commands that enable you to create, edit, and store your own segments. The two primary text editors on Multics are commands named qedx (short name "qx") and emacs. The qedx editor is a line-oriented editor; that is, you may make changes to one or more lines of text on a line by line basis. It is designed to be used on a printing terminal. There are two modes of operation in qedx: an input mode for entering new text, and an edit mode. The emacs editor is a character-oriented editor designed for video terminals. In emacs, no distinction is made between edit and input modes - you are free to edit as you type in text.

This section is a very brief introduction to both qedx and emacs. It describes the fundamentals of the commands and provides you with some of their most useful requests. However, the qedx and emacs editors have user guides that explain them fully: the emacs Text Editor Users' Guide, Order No. CH27, and the qedx Text Editor Users' Guide, Order No. CG40. One of these manuals should be among your group of essential Multics manuals.

THE qedx EDITOR -- PRINTING TERMINALS

There are two modes of operation in qedx: an input mode for entering new text, and an edit mode. In input mode, you type information into the system; that is, you create text. In edit mode, you make changes to already existing text. A set of qedx requests allows you to enter and leave input mode, perform the various editing functions (substitution, deletion, printing), and save your text in segments.

Creating Text

To enter the qedx editor, you type the qedx command, "qedx" (or "qx"), and a newline. Now you are in edit mode; notice that there is no prompt to remind you of this. In edit mode, qedx expects requests from you.

To enter input mode and create text, type "a" -- the append request -- and then a newline. Now you type in your text. The erase and kill characters still perform their correcting functions within input mode. The characters "\f" together (backslash-f) at the beginning of a line end your input session and return you to edit mode. Here is an example of creating text:

```
    r 13:17 1.010 93
!   qedx
!   a
!       Doggerel
!   My dog is Brie:
!   he follows me
!   and chews on tennis balls.
!   He's got a stick,
!   but not one tick,
!   and sometimes barks in halls.
!   \f
```

Be sure to type the "\f" as soon as you want to stop creating text. Until you type "\f", EVERYTHING you type is included as part of your text, including any requests you type thinking that you have already left input mode! So, if qedx does not respond to your editing requests, try typing "\f". When this occurs, you will probably need to delete several lines of editing requests from your text.

Editing Text

The qedx editor treats text as a series of numbered lines, the top line being line 1. The editor also renumbers the text every time lines are added or deleted. For example, if you delete line 1 of your text, then line 2 becomes line 1, and line 3 becomes line 2. Because the number of lines in your text changes often, the "$" character has been established to stand for the the last line, no matter what its number is. These line numbers are implied -- they do NOT appear next to your text.

Within the editor is an implicit "current line pointer", which is always "pointing at" one line of your text -- the last line you typed in or edited. The "=" (equal) request prints the line number of the current line, that is, where the current line pointer is located at the moment.

Another method of locating lines of text moves the current line pointer to the line you request and then prints that line. This method consists only of an address: either an absolute line number ("2", "$"), a relative line number (relative to the current line: "-1", "+3"), a character string between slashes ("/Br/"), or a period, which always prints the current line. For example:

```
! 5
  He's got a stick,
! +1
  but not one tick,
! $
  and sometimes barks in halls.
```

There is no letter associated with this method of finding lines.

Addresses are often used in conjunction with editing requests to specify on which lines changes are to be made. The request can be preceded by one address, as in this example using the "d" (delete) request to delete the line:

```
! /gg/d
```

or it can be preceded by two addresses with a comma between (but no spaces), as in this example using the "p" (print) request:

```
! 1,2p
  My dog is Brie:
  he follows me
```

Using two addresses with a request means that the request operates on the first line THROUGH the last line addressed. It is a good idea to use the "p" request often, to make sure that you are doing what you intended.

To modify words within lines, use the "s" (substitute) request preceded by one or two addresses. The "s" request expects to be followed by two character strings separated by slashes (FORWARD slashes -- not the backslash of the "\f" request!) telling it the old string and the new string:

```
! s/from old/to new/
```

If you wanted to change the word "and" to the word "And" on the third line, you might type:

```
! 1s/a/A/
```

Be careful, though - an "s" request makes this change on ALL instances of the character string you supply, on the line or lines you specify. If there is a mistake, you can use the "s" request to fix it:

```
! 3
  And chews on tennis bAlls.
! s/bA/ba/
! .
  d chews on tennis balls.
```

The default value for the "d", "p", and "s" requests is the current line.

Saving Text

When you are in the qedx editor (in either mode), the text you enter is kept in a temporary space called a buffer. All your work in this buffer is discarded when you leave qedx and return to command level (with the quit request, "q"). In order to save your work in a segment, you must use the "w" (write) request, along with a name you choose, to write a copy of your work from the buffer into a segment with the name you have provided:

```
      <text of poem>
!     w Doggerel
!     q
      r 13:29 2.749 1102
```

Now you have a permanent segment named "Doggerel". Remember: unless you type a "w" and the name of the segment, your work is not saved.

When naming your segments, keep in mind that you should use only alphabetic and numeric characters, underscores (rather than spaces), periods, and hyphens. Names can be up to 32 characters long. Remember also that uppercase characters are distinguished from lowercase characters.

When you want to return to qedx and edit an already existing segment, for example the "tadpole" segment, you must "read" a COPY of it into the editor at the beginning of your editing session. Do this at qedx request level by typing the "r" (read) request, followed by the name of your segment:

```
!     qx
!     r tadpole
```

Before you do this, the buffer is empty. Now it contains a copy of the segment "tadpole".

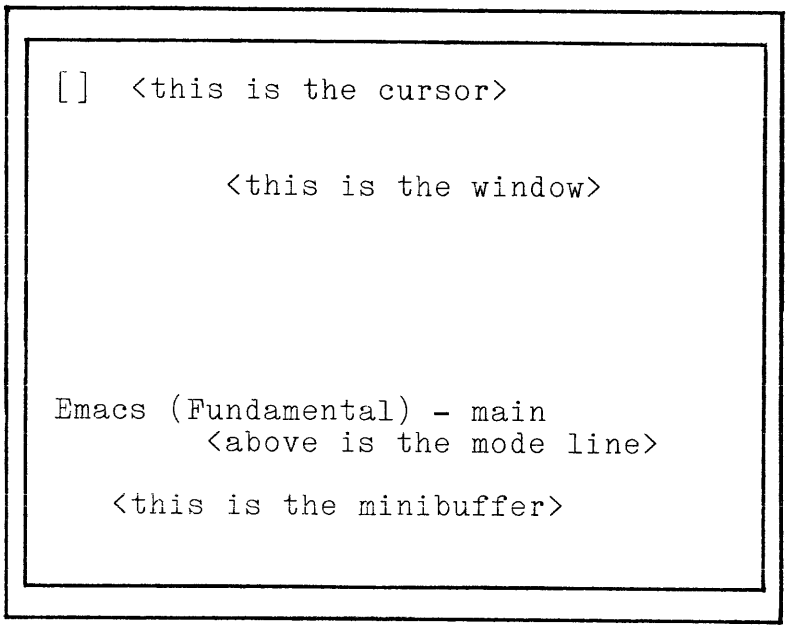
Check the qedx Text Editor Users' Guide for any questions you may have. It describes all aspects of the qedx editor in complete detail.

THE emacs EDITOR -- VIDEO TERMINALS

The emacs editor is designed especially for use at a video terminal. It is a character-oriented editor: the cursor of the terminal (the blinking underscore or square) acts as a visible pointer to one character at a time. Text is entered simply by typing. Changes can be made at any time, by typing requests that move the cursor to the point desired and then adding or deleting as you wish. A limited set of requests is described below.

Creating Text

To enter the emacs editor, you type the emacs command and a newline. Your screen clears, and you are at request level, in Emacs Fundamental mode; a banner announcing this, called the mode line, is displayed now at the bottom of your screen. The space above displays the contents of the buffer (the temporary workspace for text entry and editing) and is called a window. Below the mode line is a space called the minibuffer; messages from emacs, and your replies, are displayed here, so as not to interfere with your text. Here is a diagram illustrating a video screen after you've entered emacs:



To enter text, you simply type, as you would on a typewriter. Your text appears on the screen a few seconds after you enter it; type a carriage return whenever you want to start a new line. The lines displayed in the window are EXACTLY what your text contains.

You may erase characters and kill lines as you normally do on Multics, and then type in the correct text. Notice that while you work with emacs, the erase and kill characters actually cause the deleted material to disappear from your screen.

emacs Requests

You have a whole array of emacs requests for moving the cursor to the point where a correction is needed. The two types of requests introduced here are control characters and escape characters.

A control character, for instance a "control P", is executed with the key labeled CTL or CONTROL. You use it like the shift key, so that for a "control P" you hold down the CTL key WHILE typing an uppercase or lowercase P. A control character is expressed by a carat (^) followed by the appropriate character: ^P for "control P".

Escape characters use the key labeled ESC, ESCAPE, ALT, or ALTMODE on your terminal. Do not hold this key down while you type its associated character -- type ESC and release it, and THEN type the next character. An escape character is expressed by the letters ESC followed by the appropriate character: ESC V for "escape V".

Editing Text

In order to change some text already entered, you need to know how to position the cursor. Here is a short list of requests that move the cursor through the lines of your text. These requests operate even if the affected lines are not currently on display in the window (the window displays only about 20 lines of your text at a time).

^P (prev-line-command)
moves the cursor to the previous line, trying to stay in the same column.

^N (next-line-command)
moves the cursor to the next line, trying to stay in the column.

^V (next-screen)
displays the next screenful of text in this buffer, leaving the cursor ot the top of the screen.

ESC V (prev-screen)
displays the previous screenful of text (one back) in this buffer, leaving the cursor at the top of the screen.

ESC < (go-to-beginning-of-buffer)
moves to the first line of text in the buffer, leaving the cursor in front of the first character.

ESC > (go-to-end-of-buffer)
moves to the last line of text in the buffer, leaving the cursor just after the last character.

The following requests are useful for moving the cursor within one line:

^F (forward-char)
moves the cursor forward a character at a time.

^B (backward-char)
moves the cursor backward a character at a time.

ESC F (forward-word)
moves the cursor forward a word at a time.

ESC B (backward-word)
moves the cursor backward a word at a time.

^A (go-to-beginning-of-line)
moves the cursor in front of the beginning of a line.

^E (go-to-end-of-line)
moves the cursor to just after the last character of a line.

Once you reach the right place, you can use either the erase and kill characters or one of the following deletion requests:

^D (delete-char)
delete the character at the cursor.

ESC D (delete-word)
delete the word at and forward (to the right) of the cursor.

^K (kill-lines)
delete from the cursor to the end of that line; when at an empty line, delete that.

When you type any of these requests, the deleted text disappears from your screen, and surrounding text closes up the space. The cursor remains, and you just type in any text you'd like to add. The surrounding text moves aside as you type.

Saving Text

As noted above, when you are in the emacs editor, the text you enter is kept in a temporary space called a buffer. All your work in this buffer is discarded when you leave emacs and return to command level. In order to save your work on emacs, you must issue the "write-file" request, ^X^W, to copy your text into a segment with a name you have provided. After typing ^X^W, a message in the minibuffer prompts you for the pathname of the segment into which you want to "write", or copy, your buffer's contents. (Whenever emacs prompts you, the cursor moves into the minibuffer so you can type your reply there without interfering with your previous work.) Type the pathname and a carriage return. Now a permanent segment is created with the contents of your buffer, and the cursor returns to its previous position in your text. Note that the pathname appears in the minibuffer. At this point you can continue adding text if you want, and repeat this request later, or you can leave the editor.

To leave emacs, you type the "quit-the-editor" request, ^X^C. If you have typed new text into your buffer, or changed it in any way without writing it out, emacs asks you if you really want to quit. You have to respond "yes" if you do. If you did write your buffer out, you are immediately returned to command level.

When you want to return to emacs and edit an already existing segment, for example the "tadpole" segment, you must place a COPY of the segment into the editor's buffer at the beginning of your editing session. Do this at emacs request level by typing the "find-file" request, ^X^F, which will prompt for the name of the segment to be edited. After the segment is read in and displayed in the window, the cursor is left at the beginning of the buffer. Since your screen is small, it does not always display your whole buffer, but it is there, nevertheless.

Getting Help From Emacs

The emacs editor has an extensive online documentation system that can provide information about any request, tell you what any request does, tell you what requests you gave recently, or help you determine what request is needed for a specific task.

To find out what any given key or control character does, type ESC ? and then the key or control character you have questions about.

All other help requests center around the emacs "help" request, ^_ (control-underscore). (On some terminals, ^? performs this function; if one doesn't work, try the other). The help request may be your most useful request, as you begin experimenting with emacs. Type "^_?" to find out more about what help facilities are available.

Use the "LINEFEED" or "LF" key to remove the displays produced by these requests. If your terminal has an automatic line feed, you may need to turn off this feature to prevent these displays from vanishing from your screen as soon as they appear.

See the emacs Text Editor Users' Guide for any questions you may have. It describes, in tutorial style, all aspects of the emacs editor.

SECTION 5

SEGMENTS

A segment is a collection of information identified by a pathname. A single segment may contain any amount of information, from none to over one million printed characters.

As a new user, you will be working with ASCII(1) segments. An ASCII segment contains information that can be printed on a terminal, for instance a list of names or a section of a manual. A non-ASCII segment cannot successfully be printed on a terminal: it would appear as strings of digits and backslashes.

There are many commands on Multics that enable you to work with both your own segments and those of other users. In this section you are introduced to several commands that help you manipulate your segments.

VIEWING SEGMENTS

The print Command

One of the most common commands used with segments is the print command (short name "pr"), introduced in Section 3. Here is the syntax line again:

```
print path {begin_line} {end_line}
```

Try printing one of your segments using one or both of the optional arguments.

---

(1) The word "ASCII" is the acronym for American Standard Code for Information Interchange. This is the code used by Multics and other computer systems to represent printable text.

Another way to see your segment is to use an editing command, as you did to create your segments. For example:

```
! qedx
! r Doggerel
! 1,$p
  My dog is Brie:
  he follows me
  and chews on tennis balls.
  He's got a stick,
  but not one tick,
  and sometimes barks in halls.
! q
  r 14:23 0.119 150
```

The dprint Command

Another way to receive a printed copy (a dprint) of your segment is to use the dprint command (dp). This command orders a copy of your segment to be printed, not at your terminal, but on a high-speed printing device called a line printer. The syntax line for the dprint command is:

```
dprint {-control_args} path
```

Two useful control arguments to this command are:

-copy N, -cp N
: prints 2, 3, or 4 copies, whichever you specify in place of N. If you do not use this control argument, you receive 1 copy (in other words, 1 is the default value for N).

-destination "address", -ds "address"
: allows you to specify where the dprint is to be sent. You type the address as an argument to the -ds control argument; if you want to include blanks, you must enclose the entire address in quotation marks, as indicated above. If you do not use this control argument, the default address is your Project_id.

  Your site may require that you include the -ds control argument with a particular value, in order to aid in delivery. Check with your project administrator to see if this is required at your site.

The order in which you type arguments is significant with this command: you should type the pathname AFTER you type all the control arguments.

Here is a complete sample dprint interaction:

```
! dprint -ds "4th floor" tadpole
  1 request signalled, 2 in printer queue 3.
  r 14:26 0.442 236
```

The message sent by Multics displays the number of dprints Pam requested, and informs her of the number of requests already in line (in "queue 3") waiting to be dprinted.

Find out where you can have dprints delivered, and try dprinting a segment.

NAMING SEGMENTS

As you saw when you created segments with an editor, you provide a name for each of your segments. You may add extra names, called alternate names, to the original name of your segment with the add_name command (an). The syntax line for the add_name command is:

```
add_name path name1 {... nameN}
```

The path argument is your segment's original pathname, and name1 is an alternate name; "{... nameN}" means that you can also add as many alternate names as you wish. For example:

```
! add_name tadpole polliwog amphibian
  r 14:27 0.102 73
```

In this example, two alternate names -- "polliwog" and "amphibian" -- are added to the segment "tadpole".

It is also possible to rename a segment, with the rename command (rn). Its syntax line is:

```
rename path name
```

and it allows only one new name ("frog") to replace the old name ("tadpole"):

```
! rename tadpole frog
  r 14:29 0.061 67
```

The rename command has no effect on any alternate names your segment might have: the alternate names remain with the segment.

If you decide that one (or more) of the names of a segment in not appropriate, you may delete it with the delete_name command (dn):

```
delete_name paths
```

Here is an example:

```
! delete_name polliwog
  r 14:29 0.054 55
```

Now the segment "frog" has only one alternate name: "amphibian".

Deleting the names of a segment is quite different from deleting the segment itself. You may delete as many names as you want, as long as you do not delete every name from the segment. A segment must have at least one name though, so if you accidentally attempt to delete every name, you receive a warning message:

```
! delete_name amphibian frog
  delete_name: The operation would leave no names on
               entry. >udd>Doc>PSissle>frog
```

The first name listed on the above command line does get deleted, though.

SEGMENT ATTRIBUTES

The names of a segment are considered to be among its attributes, or characteristics. Other attributes of a segment are the author, the length, and the access modes: what permission you have to see, change, and add to your segment (access information is described in Section 7, "Access").

The list command (ls) gives you information about several attributes of your segments. The list command syntax line is:

```
list {path}
```

When you include the name of a segment on the list command line, you receive information concerning three attributes of that segment. Here is a list of the attributes of the segment "Doggerel":

```
! list Doggerel
   Segments = 1, Lengths = 1
   r w    1  Doggerel
   r 14:35 0.103 78
```

The list command response begins with a header line that gives the total number of segments and the sum of their sizes. On the next line, the first group of letters ("r w") identifies what combination of the three access modes you are permitted on the segments. Here you have permission to read (see) your segment and write (edit) it. The second attribute ("1") is the size of your segment, in pages. A page of a segment is up to or exactly 4096 characters. The third attribute is the list of names for the segment, in this case the one name "Doggerel".

When you type the list command with no arguments, you get a list like the one described above, except that it includes information about all of your segments. It also begins with a header:

```
! list

   Segments = 4, Lengths = 5

   r w     2  lizard
   r w     2  frog
   r w     1  Doggerel
           0  PSissle.mbx

   r 14:40 0.271 102
```

The last segment listed is Pam's mailbox. The absence of access attributes, and the "0" length, do not mean that you have no mail now. They signify that you cannot use your mailbox segment with any standard commands (like print); you have access only through the mail facility commands.

DELETING SEGMENTS

Finally, to get rid of a segment that you no longer need, use the delete command (dl) with the name of a command:

```
! delete frog
   r 14:42 0.305 214
```

When you delete a segment that has alternate names, they are automatically deleted too.

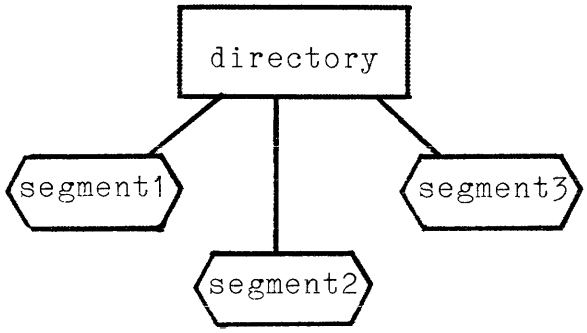
Try creating a few practice segments. Add, change, and delete names and segments, using the list command to check what you have done.

For information on seeing other people's segments, read Section 6, "Directories". To discover how to share your segments with other users, read Section 7, "Access".

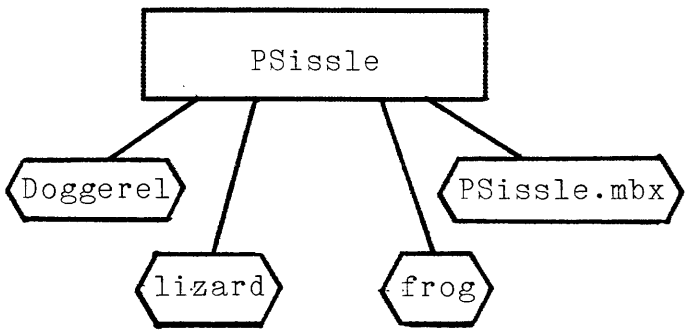
SECTION 6

DIRECTORIES

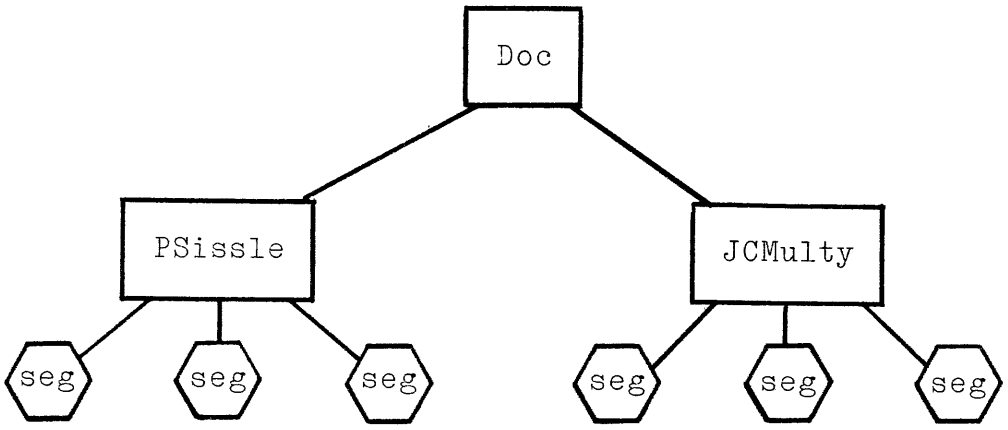
A directory is a catalogue of the segments "beneath" it. It contains information about all the attributes of each segment. Every segment and directory in the system (except the root directory, the originating directory of the Multics system) is immediately under another directory. Imagine it this way:



Without realizing it, you have been working "under" your own directory. This directory is called your home directory. It is usually named after you; its name is the same as your Person_id. You are always placed in your home directory when you log in, so when you log in, your home directory is your working directory. (You learn how to make another directory your working directory in "Changing Directories" below.) The list command can be thought of as a command that prints the contents of your home directory: the list of segments under your Person_id, and the attributes of each segment. Here is a diagram of Pam Sissle's home directory:

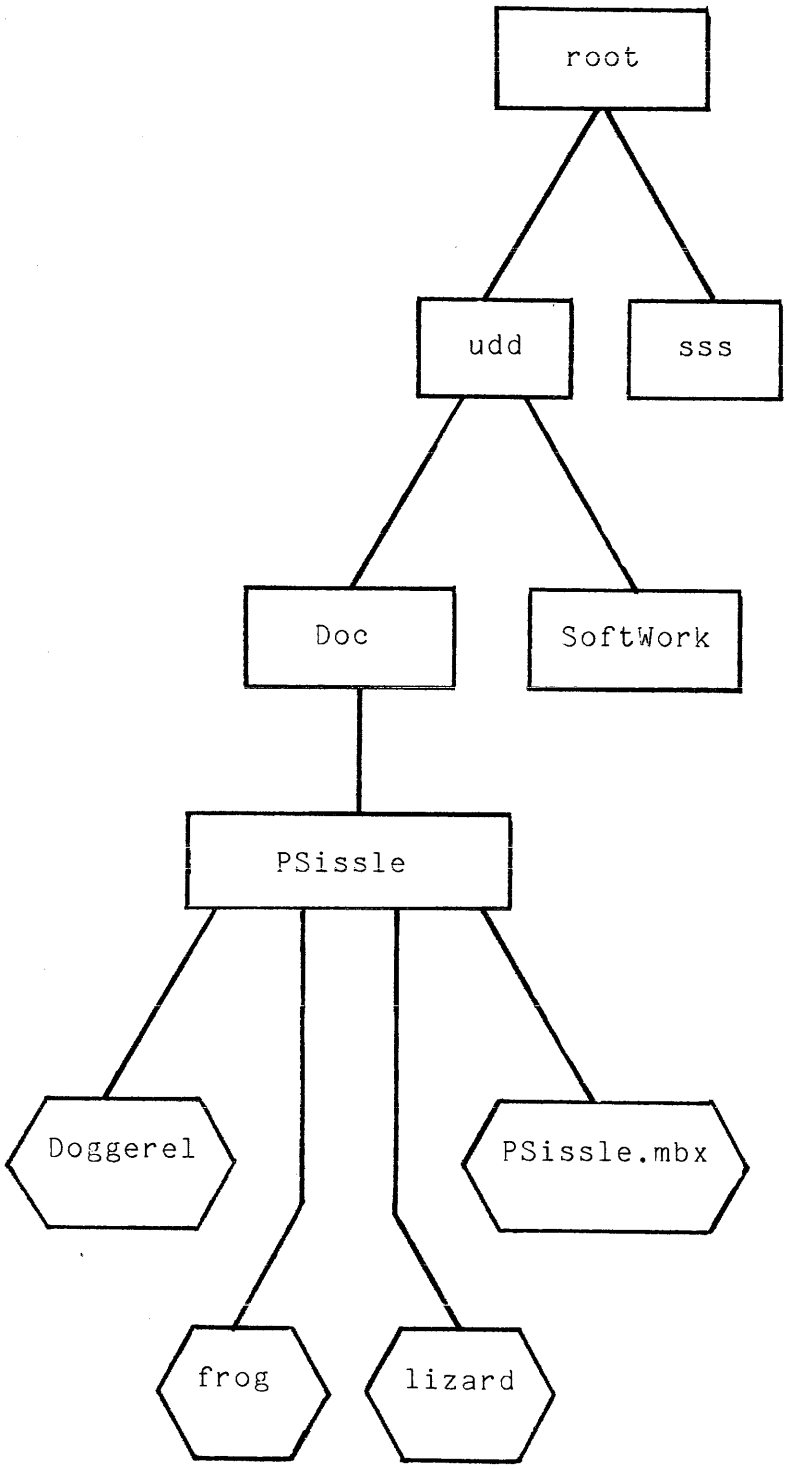


Each user in your project has a home directory similar to yours. The home directories of users are often called user directories. All of the user directories of the people in your project are grouped under another directory: the project directory. Your project directory has the same name as your Project_id. For example:



Your project is one of many projects that use your Multics system. All of the project directories are gathered under a directory called the user_directory_directory, or "udd".

The segments and directories just described are part of the Multics storage system. Its organization is analogous to the structure of an inverted tree:



The udd directory, and several additional directories governing other parts of the Multics system (not shown above), are directly beneath the root directory, which stands alone at the top of the directory hierarchy.

Absolute Pathnames

Every segment in the Multics storage system is identified by its absolute pathname. The absolute pathname of a segment is actually a series of names, separated by the ">" (greater-than) character, representing the position of that segment in this directory hierarchy. For example, the pathname:

```
>udd>Doc>PSissle>frog
```

stands for the segment named "frog", located under the user directory named "PSissle". The "PSissle" directory is stored under the project directory "Doc", which in turn is stored under the directory "udd", which is stored under the root directory (">").

When you type the absolute pathname of a segment, it always ends with the segment name. The other names in the absolute pathname are all names of directories.

Notice that the root directory is not explicitly named in the absolute pathname. Because it is present in the path of every segment, its presence is always implied by the initial ">" character of the pathname.

Relative Pathnames

The pathnames you have been using in earlier sections of this manual are called relative pathnames. Relative pathnames (which never begin with the ">" character) are shortened versions of absolute pathnames. The relative pathname of a segment begins after -- is RELATIVE to -- the name of your working directory. When you type a relative pathname, the system automatically adds on the absolute pathname of your working directory. Thus when Pam Sissle is in her home directory and types:

```
! print Doggerel
```

She sees the same segment that she would see if she typed:

```
! print >udd>Doc>PSissle>Doggerel
```

For convenience, use relative pathnames whenever you can: for all segments under your working directory.

SHARING SEGMENTS

There are many times when absolute pathnames are useful, and even essential. To print someone else's segment, you type the absolute pathname, since you do not have that segment in your own directory. Pam Sissle, from her home directory, prints Jakob Kissle's segment "neonate" with this command line:

```
! print >udd>SoftWork>JTKissle>neonate
```

To place a copy of this segment in her own directory, so that she may modify it for her own use, Pam uses the copy command (cp):

```
! copy >udd>SoftWork>JTKissle>neonate  neonate
  r 14:54 0.191 82
```

The syntax line for the copy command is:

```
copy original_path new_path
```

The first pathname is the name of the segment that is to be copied. The segment "neonate" is in Jakob's directory, so Pam specifies its absolute pathname. She wants the copy of Jakob's segment to be placed in her home directory. Because she is already in her home directory, the system automatically places the rest of the absolute pathname in front of the relative pathname.

Access to Segments

Many of the commands you have learned about in previous sections can be used with absolute pathnames, for example the dprint command and the segment naming commands (see Section 5). However, you may not have the correct access to the segments when you try to use these commands. If you do not, you will receive an error message like this one:

```
! copy >udd>Groups>JMLie>ortho eight
  copy: Incorrect access on entry. >udd>Groups>JMLie>ortho
  r 14:56 0.076 69
```

If this happens to you, try sending a message to the person whose segment you want, asking for the correct access. Also, read Section 7, "Access", for more information about access.

CREATING DIRECTORIES

It is possible to remain in your home directory and do all your work on Multics from there. It is also possible for you to create a directory underneath your home directory. You can even create your own directory hierarchy, if you like. You can then place both new and existing segments into these directories. This can be very useful as a means of organizing your segments into categories.

To create a directory, use the create_dir command (cd) with the name of the new directory:

```
! create_dir mammals
  r 14:59 0.068 69
```

You may use the create_dir command (cd) to continue building a directory hierarchy under your home directory up to 15 directories from the root.

To place existing segments under the "mammals" directory, you may use either the copy command (cp) or the move command. The copy command is used primarily for making copies of other people's segments, as shown above.

The move command (mv) is most useful for moving your own segments from your home directory to a more appropriate directory. While the copy command leaves the original segment where it is and makes a copy in the new location, the move command moves the original segment away from the first location and into the new location. The syntax line is:

```
move original_path new_path
```

For example, Pam uses the move command from her home directory:

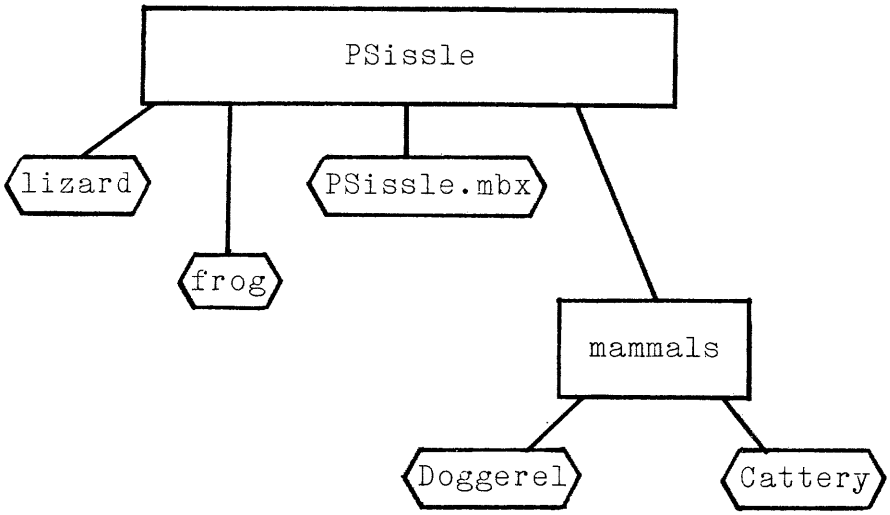
```
!  move  Doggerel  mammals>Doggerel
   r 15:01 0.489 166
```

In this example, Pam Sissle moves her "Doggerel" segment from her home directory down to her "mammals" directory, using relative pathnames (relative to her home directory) for both arguments.

To place new segments under her "mammals" directory, Pam can use an editor to specify the location she wants at the same time she creates the segment.

```
!  qedx
!  a
!  I do not have a cat.
!  What do you think of that?
!  \f
!  w mammals>Cattery
!  q
   r 15:05 0.254 256
```

Instead of writing her text with just a segment name, she gives it a longer relative pathname that places the segment named "Cattery" in the "mammals" directory, which is assumed by Multics to be underneath her home directory. Here is a diagram of Pam's current directory hierarchy:



The segment naming commands introduced in Section 5 can be used to add, change, and delete directory names also.

CHANGING DIRECTORIES

You may want to change to another directory and use it as your working directory. Your home directory has been your working directory so far. To change your working directory, use the change_wdir command with the pathname of the directory you wish to work with:

```
! change_wdir mammals
  r 15:09 0.036 67
```

When you are working in another directory, relative pathnames are relative to THAT directory. For example, when Pam is in her home directory and she wants to print the "Cattery" segment (which is located in her "mammals" directory), she types:

```
! print mammals>Cattery
```

When she moves to the "mammals" directory, she types:

```
! change_wdir mammals
  r 15:11 0.039 52
! print Cattery
```

To check on what your current working directory is, use the print_wdir command (pwd):

```
! print_wdir
  >udd>Doc>PSissle>mammals
  r 15:14 0.034 43
```

The print_wdir command always prints the absolute pathname of your working directory.

To return to your home directory, use the change_wdir command without any pathnames after it. In the example below, Pam has followed the change_wdir command with the print_wdir command, to reassure herself of her position:

```
! change_wdir
  r 15:15 0.027 40
! print_wdir
  >udd>Doc>PSissle
  15:15 0.035 42
```

The default argument of the change_wdir command is always the pathname of your home directory.

DELETING DIRECTORIES

Finally, you may delete directories as well as segments, with the delete_dir command (dd). Because the accidental deletion of a directory could cause so much pain, the system asks if you really want to take this step:

```
! delete_dir mammals
  delete_dir: Do you want to delete the directory
              >udd>Doc>PSissle>mammals??   ! yes
  r 14:57 0.434 102
```

When you delete a directory, you delete ALL the directories and segments under it, so be cautious with this command.

SECTION 7

ACCESS

The access control facility provides both flexibility and security for your work on Multics. You control who has access to each of your segments: which users have the right to see a given segment, to modify it, and, for a segment that contains a program, to execute it. Directories may also be controlled in this manner.

SEGMENT ACCESS

There are four access modes that you set to determine the level of permission a user has for a particular segment. Listed below are descriptions of the four modes, and those commands treated in this manual that a user has available with each mode.

read (r)
allows a user to read, or view, the contents of the segment

COMMANDS:

copy
dprint (dp)
move
print (pr)
qedx "r" request

write (w)
allows a user to change the contents of the segment

COMMANDS:

qedx "w" request

execute (e)
if the segment is a program, allows a user to execute the program. Execute access is not necessary at the level of this manual.

null (n)
denies a user all access to the segment

You may assign any combination of the r, w , and e modes. If you want to assign the n mode, you must assign it alone.

Every segment has its own ACL -- access control list. This list contains the full User_ids of those users who have access to the segment, and which access modes each user has been assigned. The full User_id is a three-component name that includes the tag component discussed briefly at the beginning of Section 2:

```
JTKissle.SoftWork.*
```

The tag indicates whether you are using the system interactively or as an "absentee" user (see Part II). It is usually "*", which means "any tag is acceptable". Whenever anyone tries to use any segment, that person's full User_id must match one of the entries on the ACL of that particular segment; otherwise the user has no access to that segment.

You see part of an ACL when you type the list command (ls); the access modes that you have been assigned to your own segments by default constitute your ACL entry. Here Pam Sissle checks her access to her "Doggerel" segment :

```
! list Doggerel

   Segments = 1,  Lengths = 1

   r w    1  Doggerel
   r 16:18 0.070 90
```

She has read and write permission on her segment, which allows her to do all she needs. This combination is the automatic default for most of your segments.

To see the complete ACL for her "Doggerel" segment, Pam uses the list_acl command (la) with the name of the segment:

```
! list_acl Doggerel
  r w   PSissle.Doc.*
  r w   *.SysDaemon.*
  r 16:20 0.084 89
```

Until you have added more ACL entries yourself, there are two default entries for every segment you create. The first one is your own (the one you see when you use the list command), giving read and write access to you when you log in with the given Project_id. The second is for the system daemons, software facilities that perform such functions as dprinting and saving backup copies of segments. Other users have null access by default.

To grant other users access, you use the set_acl command (sa). The syntax line for this command is:

```
set_acl  path  access_modes  User_id
```

Here Pam gives Jakob Kissle read access to "Doggerel", because he wants to make a dprint of it:

```
! set_acl  Doggerel  r  JTKissle.SoftWork
  r 16:21 0.047 42
```

(If you are registered on more than one project, you may want to grant access to your other User_ids!)

If Pam wants to let Jakob add his own verses to Doggerel, she must give him additional access. To change his access, she simply resets the ACL with the set_acl command:

```
! set_acl  Doggerel  rw  JTKissle.SoftWork
  r 16:21 0.037 45
```

Now when Pam checks the ACL on "Doggerel" this is what she sees:

```
! list_acl Doggerel
   r w   PSissle.Doc.*
   r w   JTKissle.SoftWork.*
   r w   *.SysDaemon.*
   r 16:23 0.064 83
```

But when Jakob tries to add an alternate name to the segment, all he gets is an error message:

```
! add_name Doggerel wag_tales
  add_name: Incorrect access to directory containing
            entry.  >Doc>PSissle>Doggerel
  r 16:25 0.093 92
```

He has tried to change an ATTRIBUTE of the segment. The access modes for segments govern the CONTENTS of segments. Segment attributes are stored in the directory containing the segments, so Pam must also give Jakob appropriate access to her directory, if she wants to allow him to add alternate names.

DIRECTORY ACCESS

These are the four access modes for directories. Notice that all the affected commands govern attributes of segments (names and access modes).

status (s)
: allows a user to see the ATTRIBUTES of the existing segments under the specified directory

  COMMANDS:

  list (ls)
  list_acl (la)

modify (m)
allows a user to modify attributes of the existing segments under the specified directory

COMMANDS:

add_name	(an)
delete	(dl)
delete_acl	(da)
delete_name	(dn)
rename	(rn)
set_acl	(sa)

append (a)
allows a user to create, copy, or move segments under the specified directory

COMMANDS:

copy	(cp)
create_dir	(cd)
move	(mv)

null (n)
denies a user all access to the directory. May not be assigned in combination with any of the other directory access modes.

As with the segment access modes, both you and the system daemons have complete access (sma) to all your segments and directories by default.

Use the set_acl command (sa) again to set the ACL for directories. Aside from using s, m, and a combinations rather than r, e, and w, there is one quirk you may take advantage of when setting access for directories: when setting access on your working directory, you may use "-wd" in place of the directory's pathname. For example, here Pam gives Jakob access to her directory so that he may add alternate names to the segments in that directory:

```
! set_acl -wd sma JTKissle.SoftWork
```

Of course, Jakob is now free to do other things in the directory too, such as changing the access on the segments underneath it, or even deleting them. It is often a good idea to think carefully before granting more than "r" and "s" access to other users.

To see who now has access to her directory, Pam uses the list_acl command (la) with the -wd argument:

```
! list_acl -wd
   sma   PSissle.Doc.*
   sma   JTKissle.SoftWork.*
   sma   *.SysDaemon.*
   r 16:51 0.040 77
```

Finally, to delete the access of a particular user, use the delete_acl command (da). Its syntax line is:

```
delete_acl path User_id
```

For example, here Pam has decided not to let Jakob have access to her directory (the -wd argument can be used with all the access commands):

```
! delete_acl -wd JTKissle.SoftWork
  r 16:56 0.034 45
```

Jakob still maintains the segment access that Pam gave him, though. If Pam decides to delete his access to the "Doggerel" segment, she uses the delete_acl command with the name of the segment:

```
! delete_acl Doggerel JTKissle.SoftWork
  r 16:58 0.038 41
```

The access control facility is much more extensive than this brief introduction has suggested. For a complete discussion of access, see the descriptions of these commands in the MPM Commands manual.

APPENDIX A

GLOSSARY

absolute pathname
: see pathname, absolute

access control
: a system of permission assigned by users for their own segments and directories, determining who may see, change, and add to each one. See Section 7, "Access".

access modes
: identify the types of access that may be set for a segment or directory. The access modes 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). See "Segment Access" in Section 7.

ACL, access control list
: a list of the users who have access to a particular segment or directory, and a description (using access modes) of the degree of access allowed to each user. See "Segment Access" in Section 7.

address
: within the qedx command, a character or character string that identifies one line of text. See "The qedx Editor" in Section 4.

alternate name
: an additional name for a segment or directory, assigned by the add_name command. A segment or directory can be referred to equally well by its original name or any of its alternate names. See "Naming Segments" in Section 5.

argument
: information included with a command on a command line. See "Argument Conventions" in Section 3.

ASCII
: acronym for American Standard Code for Information Interchange, the standard code used to represent printable text on Multics. With it, all the characters normally found on a typewriter keyboard, plus many special characters, may be represented. See Section 5, "Segments".

attribute
: a characteristic of a segment. A segment has many attributes, such as its names, the author, its ACL, and its size; the attributes can be examined with the list command. See "Segment Attributes" in Section 5.

banner
: one or more lines, in a standard format, that begin or comprise certain computer messages; also called a header. See "Logging In" in Section 2.

buffer
: within a text editing command (qedx or emacs), a temporary workspace in which you create and edit text. To store that work in a segment you must "write it out" (make a permanent copy) using the qedx "w" (write) request or the emacs ^X^W request. See Section 4, "Creating and Editing Text".

character string
: one group of characters unbroken by blanks; signifies one word to Multics. The characters may include alphabetic, numeric, and some other characters (periods, hyphens, and underscores). See "Command Names" in Section 3.

command
: an instruction to Multics; the command name. See Section 3, "Basic Multics Commands".

command level
: the state the system is in when it is ready to accept a command from you. See also "request level". See "Logging In" in Section 2.

command line
: one complete instruction to Multics, including a command name, arguments to that command, if any, and a newline. See "Command Lines" in Section 3.

control argument
: an optional command argument that begins with a hyphen. Control arguments are typed exactly as indicated in command descriptions. See "Login/Logout Variations" in Section 2.

crash
: an unplanned termination of computer operation, due to hardware or software difficulties.

cursor
: the blinking square or underscore that marks your current position on a video terminal. See "Login Problems" in Section 2.

daemon
: one of several system facilities that perform such services as dprinting segments on a line printer (IO.SysDaemon is the User_id for this facility) and saving copies of segments as a means of backup (Dumper.SysDaemon). See "Segment Access" in Section 7.

default
: the value or action that the system assumes when none has been specified by the user. See "Default Arguments" in Section 3.

directory
: a "catalogue" containing the names and other attributes of all segments and directories immediately beneath it in the storage system (the directory hierarchy). See Section 6, "Directories".

directory, home
: the directory that you work from when you first log in. The pathname for this directory is usually of the form >udd>Project_id>Person_id. See Section 6, "Directories".

directory, root
: see root directory.

directory, working
: the directory that you are currently working from. Your home directory is always the working directory when you log in; to move to another directory, use the change_wdir command (cwd) with the desired pathname. See Section 6, "Directories".

directory hierarchy
: the organizational structure of the Multics storage system. It resembles an inverted tree, with the root directory at the top, and subordinate directories emanating downward as branches. The lowest levels of each branch, the "leaves", are segments. See Section 6, "Directories".

dprint (for daemon print)
the command to print segments on a line printer rather than a terminal. The printed segments themselves are referred to as dprints. See "The dprint Command" in Section 5.

edit mode
a mode of operation within a few interactive commands (qedx, send_mail) in which the command is ready to accept editing requests from you. See also "request level".

erase character
the "#" character; it erases the character, or continuous series of blanks or tabs, immediately preceding it. For example, both

"tha#e"

and

"tha e###e"

are seen by Multics as "the". See "Multics Conventions" in Section 1.

error message
a message sent by Multics informing you that a command you typed was not carried out, and indicating what was wrong. For example, "Segment ! not found." is returned if you mistakenly type the exclamation point before a command. See "Login Problems" in Section 2.

hardware
the physical devices and electronic circuitry that comprise a computer. See "The Multics System" in Section 1.

hardwired
connected directly and permanently to the computer; usually refers to terminals. See "Connecting The Terminal" in Section 2.

header
see banner

help file
see info segment.

home directory
see directory, home

info segment
a segment containing information about a given command. Typing the help command, followed with the name of a command, prints the info segment for that command at your terminal. See "The help Command" in Section 3.

input mode
a mode of operation within many interactive commands (for example, qedx or send_message) in which the command is ready to accept text from you. Input mode is usually terminated by you with a special character (described in individual command descriptions, Appendix B). See "Sending Messages" in Section 3.

interactive
a mode of operation in which you and the computer exchange information directly and immediately, through a terminal. Multics is primarily an interactive system. See "The Multics System" in Section 1.

interactive command
a command that communicates interactively with you: the command expects requests or other appropriate information from you (such as a password or the text of a message), and responds to you with messages or other prompts. Examples of interactive commands are qedx, send_message, help and print_mail. See "Sending Messages" in Section 3.

kill character
the "@" character; it deletes all previously-typed characters on the current line. For example,

"Itdellet esall@ It"

becomes

" It"

See "Multics Conventions" in Section 1.

logging in
Entering the Multics system. This can be accomplished with the login command. The login command is an interactive command. See Section 2, "Entering and Leaving Multics".

logging out
Leaving the Multics system (with the logout command). See Section 2, "Entering and Leaving Multics".

mailbox
: a specially protected segment used to store messages and mail from other users. Typically, each person has a mailbox named Person_id.mbx under the home directory. A permanent mailbox is created for you the first time you type the accept_messages command (am) or the print_mail command (prm). See "Communicating With Other Users" in Section 3.

memory
: the computer hardware in which all segments are stored. See "User Identification" in Section 2.

modem
: the machinery that transmits electronic signals between a terminal, a telephone, and the computer. See "Connecting The Terminal" in Section 2.

newline
: the newline, caused by typing the key labeled "RETURN" (or another key designated for this purpose, such as "CR" or "LF"), consists of two actions: a carriage return and a linefeed. It is the signal to Multics that a command line is ready to be processed. See "Command Lines" in Section 3.

operating system
: the system software that supervises the processing of commands, controls system hardware, and organizes the tasks to be accomplished. See "The Multics System" in Section 2.

page
: the smallest unit of storage that is allocated to segments. One page contains up to 4096 characters. See "Segment Attributes" in Section 5.

password
: A character string known only to you. The system knows only an encoded form of your password. A password is assigned to you when you are registered on the system. The password is used when you log in, along with the Person_id, to verify your identity. See "User Identification" in Section 2.

pathname
: a name of a segment or directory that specifies its location in the storage system. A pathname is either absolute or relative. See "Pathname Arguments" in Section 3 and "Pathnames" in Section 6.

pathname, absolute
: a segment or directory name preceded by the series of directory names that lead from the root to that segment; each level in the pathname is preceded by a ">". For example, the absolute pathname for a segment under a user's home directory is designated this way:

```
>udd>Project_id>Person_id>segment_name
```

All absolute pathnames begin with ">". See "Absolute Pathnames" in Section 6.

pathname, relative
: the pathname that uniquely locates a segment or directory relative to your working directory, by listing only that part of the absolute pathname that comes after the name of the working directory. For example, the relative pathname for a segment that resides in a directory one level under the working directory is designated this way:

```
lower_dir>segment_name
```

All relative pathnames begin WITHOUT ">". See "Relative Pathnames" in Section 6.

Person_id
: A unique user identification; the first component of the User_id. See "User Identification" in Section 2.

project administrator
: Within this manual, this term is used for those people who have special access privileges that enable them to register new users, control the allocation of their system resources, and perform other administrative functions. At your site these functions may be shared among several people, perhaps having several different titles (system administrator and registrar, for instance). See "Login Problems" in Section 2.

Project_id
: The second component of the User_id; used for accounting purposes. See "User Identification" in Section 2.

process
: one session with Multics, usually from login to logout. For the user, this means one terminal session.

QUIT signal
a signal to Multics that you want to interrupt its current action. To send this signal you press the key labeled "ATTN", "BREAK", or "INTERRUPT"; the system responds with a QUIT banner and a ready message from a new command level. Type "release -all" to return to the first command level. The QUIT signal has nothing to do with the "quit" requests of interactive commands. See "Stopping A Command" in Section 3.

ready message
the signal that Multics is at command level, ready for the next command. Ready messages may differ from site to site, but often include information such as the current time and date, and some measure of total system resource usage. See "Logging In" in Section 2.

relative pathname
see pathname, relative

request
within an interactive command, an instruction to the command, given at request level; the requests of an interactive command are analogous to the commands of the Multics system. See "Sending Messages" in Section 3.

request level
the state in which an interactive command (for example qedx or send_mail) is ready to receive either appropriate information (text, a message) or a request from the command's list of accepted requests. Within the qedx command, request level is called "edit mode". See "Sending Messages" in Section 3.

root directory
the directory at the top of the Multics storage system; all other directories are subordinate to it. In absolute pathnames it is referred to by the initial ">". See Section 6, "Directories".

segment
a unit of information (for example, text or a computer program), which is created and named by a user, and stored in the storage system. Each segment has a set of attributes, such as its primary and alternate names and its ACL. See Section 5, "Segments".

short name
the shortened version of a command or control argument name, used interchangeably with the full name. The short names are listed in individual command descriptions (Appendix B). See "Command Names" in Section 3.

software
the programs that control the activities of the computer. See "The Multics System" in Section 1.

storage system
the combination of hardware and software that Multics uses for storing information. The information is grouped into segments and catalogued in directories. See also "directory hierarchy". See Section 6, "Directories".

syntax line
in command descriptions, the line that demonstrates the structure of a complete command line for that particular command. See "Argument Conventions" in Section 3.

SysDaemon
see daemon

tag
see User_id

terminal
a typewriter-like piece of hardware that allows a user to interact directly with the computer system. See "The Multics System" in Section 1 and "Login Problems" in Section 2.

text editor
a very powerful interactive command that allows you to enter text into the system and to edit it. See Section 4, "Creating and Editing Text".

timesharing
the ability of the computer system to allow many users, with tasks of all sizes, to use the system simultaneously. See "The Multics System" in Section 1.

User_id
A unique three-component name assigned to each Multics user. The User_id discussed in this manual consists most often of the first two components, separated by a period:

Person_id.Project_id

The third component, the "tag", is discussed in Section 7. See also "User Identification" in Section 2.

working directory
see directory, working.

APPENDIX B

COMMAND DESCRIPTIONS

This appendix contains abbreviated descriptions of the commands and arguments discussed in this manual. For complete descriptions of these and other commands, see the MPM Commands manual. The "help" command is another useful source of information, especially while you are at your terminal.

accept_messages, am

Allows incoming messages to be printed at the user's terminal.

SYNTAX: am {-control_arg}

ARGUMENT:

-print, -pr

this control argument prints all deferred messages: those that were received since the user last typed the defer_messages command or logged out. If -print is not given, the default is to print only those messages that are sent after this accept_messages command is typed.

add_name, an

Adds alternate names to the existing names of a segment.

SYNTAX: an path name1 { ... nameN}

ARGUMENTS:

path
is the pathname of the segment or directory to which the alternate names are added.

names
are the alternate names to be added to the segment or directory.

---

calendar

Prints a one-month calendar.

SYNTAX: calendar {date}

ARGUMENT:

date
is a date, of the form MM/DD/YY, that identifies which month is to be printed. If date is not given, the default is to print a calendar for the current month.

---

change_wdir, cwd

Changes the user's working directory to the directory specified.

SYNTAX: cwd {path}

ARGUMENT:

path
is the pathname of the directory that is to become the working directory. If path is not given, the home directory is assumed.

---

copy, cp

Creates a copy of the specified segment in the specified directory with the specified name.

SYNTAX: cp original_path new_path

ARGUMENTS:

original_path
is the pathname of the segment to be copied.

new_path
is the pathname of the copy. The user may choose the segment name of the copy when typing this pathname.

---

create_dir, cd

Creates a directory at a specified location in the directory hierarchy.

SYNTAX: cd path

ARGUMENT:

path
is the pathname of the directory to be created.

---

defer_messages, dm

Prevents messages from being printed at the user's terminal. (See also the accept_messages command.)

SYNTAX: dm

---

delete, dl

Deletes the specified segment.

SYNTAX: dl path

ARGUMENT:

path
is the pathname of the segment to be deleted.

---

delete_acl, da

Deletes entries from the ACL (access control list) of a segment or a directory.

SYNTAX: da path User_id

ARGUMENTS:

path
is the pathname of the segment or directory for which access is to be removed. This argument may be "-wd" if you want to delete access from the working directory.

User_id
is the User_id of the person who is to be removed from the ACL.

---

delete_dir, dd

Deletes the specified directory and its subtree.

SYNTAX: dd path

ARGUMENT:

path
is the pathname of the directory to be deleted.

---

delete_name, dn

Deletes specified names from a segment or directory.

SYNTAX: dn paths

ARGUMENT:

paths
are the pathnames to be deleted.

---

dprint, dp

Prints the specified segment on a high-speed line printer.

SYNTAX: dp {-control_args} path

ARGUMENTS:

-copy N, -cp N
this control argument prints 2, 3, or 4 copies of the specified segment, according to the number supplied by the user. If -copy is not given, the default is 1 copy.

-destination "address", -ds "address"
This control argument specifies where the dprint is to be sent. To include blanks in the address, enclose the entire address in quotation marks; for a single character string this is unnecessary. If -ds is not given, the default is the user's Project_id.

All control arguments must be typed before the path argument.

path
is the pathname of the segment to be dprinted.

---

emacs

Creates and edits text segments.

Refer to Section 4 for command description.

---

help

Prints blocks of information (info segments) about how to use the specified command.

SYNTAX: help {command_name} {-control_args}

ARGUMENTS:

command_name
is the name of a Multics command. If no command name is given, the default is the help command info segment.

-all, -a
this control argument prints the entire info segment without prompting the user after each block. If -all is not given, the default is to prompt the user for a request after each block of information.

-brief, -bf
this control argument prints only a one-block summary of the command information. If -brief is not given, the default is to begin printing the standard set of information blocks, each one followed by a prompt.

REQUESTS:

quit
stops printing information, and returns the user to command level.

rest
prints the rest of the info segment, and returns the user to command level.

skip
skips one information block.

yes
prints the next information block.

---

list, ls

Prints information about specified segments and directories.

SYNTAX: ls {path} {-control_arg}

ARGUMENTS:

path
is the pathname of the segment or directory to be listed. The pathname of a segment is used without the -dir control argument to print the size, all names, and access information about that segment. The pathname of a directory is used with the -dir control argument to print access and name

-directory, -dir, -dr
this control argument, when used without the path argument, prints access and name information about all directories located immediately beneath the working directory.

When no arguments are given with the list command, the default is to print information about all segments immediately beneath the working directory.

---

list_acl, la

Lists the ACL (access control list) of a segment or directory.

SYNTAX: la path

ARGUMENT:

path
is the pathname of the segment or directory for which the ACL is to be listed. This argument may be "-wd" if you want to list the ACL of the working directory.

---

list_help, lh

Prints a list of the info segments that pertain to the specified topic.

SYNTAX: lh topic {-control_arg}

ARGUMENTS:

topic
is the topic to be searched for.

-all, -a
this control argument prints the names of all the info segments. If -all is not given, the default is to print the names of only those info segments that pertain to the topic given.

---

login, l

Identifies the user and grants access to the system.

SYNTAX: l Person_id Project_id {-control_args}

ARGUMENTS:

Person_id
is the Person_id of the user logging in.

Project_id
is the Project_id of the user logging in. This argument is optional under some circumstances; check with your project administrator.

-brief
this control argument suppresses the standard set of messages associated with a successful login. If -brief is not given, the default is to print these messages as soon as the user types the correct password.

-change_password, -cpw
this control argument changes the user's password to a newly given password of the user's choice (not longer than eight characters). The user is prompted for both the old and the new password after this control argument is given; the user should NOT type the new password on the command line.

---

logout

Terminates a user session and breaks the terminal's connection to the computer.

SYNTAX: logout {-control_args}

ARGUMENTS:

-brief, -bf
this control argument suppresses the standard logout messages. If -brief is not given, the default is to print the logout message.

-hold, -hd
this control argument terminates the user's session but retains the terminal connection, allowing another user to log in without redialing the modem; the initial Multics banner (greeting message) is printed. If -hold is not given, the default is to end both the user session and the terminal connection, and print the message "hangup". If both -hold and -brief are given, neither the logout message nor the initial Multics banner are printed, and only a newline signals the user to log in.

---

move

Moves a segment to the specified directory with a specified name.

SYNTAX: move original_path new_path

ARGUMENTS:

original_path
is the pathname of the segment to be moved.

new_path
is the pathname of the moved segment. The user may choose the name of the moved segment when typing this pathname.

---

print, pr
Prints a specified segment on the user's terminal.

SYNTAX: pr path {begin_line} {end_line}

ARGUMENTS:

path
is the pathname of the segment to be printed.

begin_line
is the line number on which printing begins. If no number is given, the default is to begin printing from line 1.

end_line
is the line number on which printing ends. If no line number is given, the default is to stop printing after the last line of the segment. The end_line argument can only be used when the begin_line argument is also used.

---

print_mail, prm
Prints all messages residing in the user's mailbox, and prompts for requests about deletion after each message.

SYNTAX: prm {-control_arg}

ARGUMENT:

-list, -ls
this control argument prints a summary of the messages in the mailbox, before printing the messages. If -list is not given, the default is to begin printing the messages immediately.

REQUESTS:

abort
retains all current messages and returns the user to command level.

no
retains the preceding message and prints the next one.

quit
retains the preceding message and any subsequent (unprinted) messages, and returns the user to command level.

reprint
repeats the preceding message and the prompting message.

yes
deletes the preceding message and prints the next one.

---

print_wdir, pwd
Prints the pathname of the user's current working directory.

SYNTAX: pwd

---

qedx, qx
Creates and edits text segments.

Refer to Section 4 for command description.

---

release, rl
Returns the user to the previously held command level; typed after the user sends a QUIT signal.

SYNTAX: rl {-control_arg}

ARGUMENT:

-all, -a
this control argument returns the user to command level 1. If -all is not given, the default is to return the user to the most recently held command level.

---

rename, rn

Renames the specified segment or directory with the specified name.

SYNTAX: rn path name

ARGUMENTS:

path
is the pathname of the segment or directory to be renamed.

name
is the new name of the segment or directory. Remember that no two segments or directories that reside immediately under the same directory can have the same name.

---

send_mail, sdm

Sends mail to another person registered on the system; the send_mail command prompts for the User_id of the recipient.

SYNTAX: sdm User_id

REQUESTS:

print
prints the existing message.

quit
leaves the send_mail request level and returns the user to command level.

send
sends the existing message to the mailbox of the user specified on the command line, along with an interactive message that prints "You have mail."

---

send_message, sm

Sends one or more interactive messages to another user.

SYNTAX: sm User_id {message}

ARGUMENTS:

User_id
is the User_id of the recipient, in the form "Project_id.Person_id". Note the period between the recipient's Person_id and Project_id.

message
is the one-line message to be sent. If message is not given on the command line, the default is to print the prompting message "Input:" and a newline. The user may then type as many one-line messages as desired; each line is sent immediately.

Type a period (.) to terminate message input and return to command level.

---

set_acl, sa

Modifies the ACL (access control list) of a segment or directory.

SYNTAX: sa path access_modes User_id

ARGUMENTS:

path
is the pathname of the segment or directory for which access is to be set. This argument may be "-wd" if you want to set the ACL on the working directory.

access_modes
are the access modes you wish to set for the given user on the given segment. Access modes are:

SEGMENTS:		DIRECTORIES:	
read	(r)	status	(s)
write	(w)	modify	(m)
execute	(e)	append	(a)
null	(n)	null	(n)

User_id
is the User_id of the person whose ACL entry you wish to add or change.

---

who

Prints a header similar to the initial banner seen at login, and lists the User_ids of the people currently logged in to the system.

SYNTAX: who {-control_arg}

ARGUMENT:

-brief, -bf
this control argument suppresses printing of the header.

INDEX

Q

R

S

T

U

W

HONEYWELL INFORMATION SYSTEMS
Technical Publications Remarks Form

TITLE: SERIES 60 (LEVEL 68)
NEW USERS' INTRODUCTION TO MULTICS
PART I

ORDER NO. CH24-00

DATED NOVEMBER 1979

ERRORS IN PUBLICATION

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Your comments will be investigated by appropriate technical personnel and action will be taken as required. Receipt of all forms will be acknowledged; however, if you require a detailed reply, check here. ☐

FROM: NAME _____ DATE _____

TITLE _____

COMPANY _____

ADDRESS _____

_____

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

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 39531 WALTHAM, MA02154

POSTAGE WILL BE PAID BY ADDRESSEE

HONEYWELL INFORMATION SYSTEMS
200 SMITH STREET
WALTHAM, MA 02154

ATTN: PUBLICATIONS, MS486

Honeywell

Together, we can find the answers.

Honeywell

Honeywell Information Systems
U.S.A.: 200 Smith St., MS 486, Waltham, MA 02154
Canada: 155 Gordon Baker Rd., Willowdale, ON M2H 3N7
U.K.: Great West Rd., Brentford, Middlesex TW8 9DH **Italy**: 32 Via Pirelli, 20124 Milano
Mexico: Avenida Nuevo Leon 250, Mexico 11, D.F. **Japan**: 2-2 Jinbou-Cho Kanda, Chiyoda-Ku Tokyo
Australia: 124 Walker St., North Sydney, N.S.W. 2060 **S.E. Asia**: Mandarin Plaza, Tsimshatsui East, H.K.

33375, 7.5C1281, Printed in U.S.A.

CH24-00