HP 3000 Series 9X8LX Computer Systems

Understanding Your System

HP Part No. B3813-90001
Printed in USA 1994

First Edition
E0494

Hewlett-Packard Company
3000 Hanover Street
Palo Alto, CA 94304 U.S.A.

Printing History

The following table lists the printings of this document, together with the respective release dates for each edition. The software version indicates the version of the software product at the time this document was issued. Many product releases do not require changes to the document. Therefore, do not expect a one-to-one correspondence between product releases and document editions.

Edition	Date	Software Version
First Edition	April 1994	B.50.00

Introducing the HP 3000 Series 9X8LX

Hewlett-Packard's HP 3000 Series 9X8LX computer system for the multi-user environment combines office computing with mainframe power. The system is designed to be used in a standard office setting without the stringent environmental controls of the typical computer room. Its compact size accommodates a number of office configurations.

Typically, the system consists of the following components:

- The main computer that fits comfortably alongside a desk or table.
- The system console connected to the computer for system activities.
- An optional cabinet containing additional disk and tape drives for extra data storage.
- The data communication and terminal controller, also called the DTC, that enables you to connect multiple terminals and printers to the system.
- The external uninterruptible power system, also called the UPS, that can provide up to 15 minutes of battery backup power to the computer system.
- Terminals or PCs set up on each user's desktop and connected to the DTC.
- Various styles of printers conveniently located for easy access by several users.

In This Book

You may conclude one day that there are good reasons for knowing a little more about what goes on inside your computer. You may wonder:

- What really happens when you use your MPE/iX computer—what makes it work?
- Why you must do certain tasks, or why you must do them in a certain way.
- Whether it is time to investigate more of the full potential of your computer.

This book will *not* turn you into a computer expert. Instead, it provides an introduction to what computers do, and how they do it.

> *Worth Knowing*
>
> This book is not required reading. There is no requirement to memorize anything here, although there are ideas *Worth Knowing*, as well as facts *Worth Remembering*. What you will find are facts that you may want to know as you work with your computer and as you begin to discover its potential.

Each chapter presents a set of related ideas and facts about your computer. No chapter is very long. You may read the chapters in any order. But as with any body of knowledge, some facts and ideas are basic, while others are more advanced. Becoming comfortable with the basics will give you an advantage with everything that comes later.

Chapter 1: What Is A Computer?

The parts of computer, including the hardware and the programs that make it run.

Chapter 2: Where Am I?

When log on, where do you find yourself inside the computer's account structure? Knowing the account structure, and how files are named, will help to guide you to the work you want to do. The meaning of user names, session names and passwords.

Chapter 3: What Are Files?

The nature and structure of files—where they are kept. How the computer uses files to move information from one place to another.

Chapter 4: Here I Am—What Can I Do?

How your user capabilities affect your work. Programs and how to start them running on your computer.

Chapter 5: Where Does the Information Go?

How the computer gets information from you, and what it does with that information. Protecting your information once it is in the computer.

Chapter 6: Behind the Scenes

How the computer sees things.

Chapter 7: **Commands**

Types of commands, using parameters, understanding how to read the syntax diagram of a command.

Chapter 8: **Command Files and Jobs**

You create them for your own use—command files and jobs.

Chapter 9: **Jobs and Job Files**

You create them and tell the computer how and when to do your work for you.

Related Manuals

Understanding Your System is the first book in a set of five manuals that includes the following:

Understanding Your System (B3813-90001)	If you are new to computers, this is a good place to start. It provides an introduction to what computers do and how they do it.
Getting Started (B3813-90003)	Familiarizes you with your computer and computer peripherals. It also explains how to get your system ready for use, how to use and maintain your tape drives and how to communicate with your system using HP Easytime/iX.
Task Reference (B3813-90009)	Describes how to communicate with your system using MPE/iX commands. This book also includes a chapter on how to get more information on your system, a chapter on troubleshooting and a glossary.
Commands Reference (B3813-90011)	Provides a detailed explanation of each MPE/iX command.
New Features of MPE/iX: Using the Hierarchical File System (32650-90351)	Describes the changes to MPE/iX as of Release 4.5 and 5.0, which enhanced MPE/iX to make it "POSIX compatible." The book also describes the features of the hierarchical file system.

Note

The MPE/iX operating system has been enhanced as of Release 4.5 and 5.0 to include additional features that include POSIX compatibility and the *hierarchical file system*. The *hierarchical file system* is tree structured and can contain files at many different levels. This organization provides a special kind of file called a **directory**. Instead of holding data, directories contain lists of files and pointers to those files.

For more information on *POSIX* and the *hierarchical file system*, refer to the book, *New Features of MPE/iX: Using the Hierarchical File System* (32650-90351), included in this documentation set. This book includes an overview of the following enhancements of MPE/iX as of Release 4.5 and 5.0:

- Open systems environment
- Hierarchical file system (HFS)
- Expanded file naming syntax
- New and enhanced commands and utilities
- MPE/iX Shell and Utilities
- MPE/iX Developer's Kit

Caution Symbol

The Caution symbol calls attention to an operation or installation procedure, practice, or the like, that, if not correctly performed or adhered to, could result in damage to or destruction of part or all of the product. Do not proceed beyond a Caution symbol until the indicated conditions are fully understood and met.

Contents

1. What Is a Computer?

Parts and Pieces—Seen and Unseen 1-2
Hardware 1-2
Peripheral devices 1-3
Software 1-4
Files 1-6
What Are Programs? 1-8
How Do Programs Do What They Do? 1-10
The Idea Behind Computing 1-11
Your mental computer 1-12
So a computer is.... 1-13
What computers do 1-14
Interruptions 1-14
Shutdowns 1-14
Backups 1-15
System too busy 1-15
What to do 1-15

2. Where Am I? Logging On

Maintaining Order 2-2
Gaining Admission 2-7
Starting a Session 2-8
When you log on 2-9
I have logged on—now where am i? 2-9
The address of files 2-10
The full names of files 2-10
Where is that file? 2-12
LISTFILE options 2-13
Session names 2-14
Password protection 2-15

Logging on without a group 2-16
Shared group 2-17
SYS—A Special Account 2-18
MANAGER.SYS 2-18
OPERATOR.SYS 2-18
The PUB group 2-18
The account manager 2-19

3. What Are Files?
Where Are Files? 3-2
Disk memory 3-2
File recording 3-3
Pointing to the right pieces 3-3
The naming of files 3-4
The Location of Files 3-7
File names and logons 3-8
Types of files 3-8
Execution or information 3-9
Mnemonics 3-9
The File Connection 3-10
Devices 3-10
$STDIN and $STDLIST 3-11
The LDEV connection 3-12
The console 3-13
Other LDEVs 3-13
The file equation connection 3-14
Input default for PRINT 3-17
Redirecting STORE 3-18
Configuration—teaching the computer 3-19
Spool files/print files 3-20

4. Here I Am—What Can I Do?
Capabilities 4-2
Who Can, Who Cannot 4-3
In charge of the entire system 4-4
In charge, day to day 4-4
In charge of an account 4-5
At the group level 4-6

Users 4-7
If you log on in 4-8
User JOHN logs on in PUB 4-8
User JOHN logs on in MYGROUP 4-9
User John logs on in OTHERGRP 4-9
Other capabilities 4-9
Access control definitions (ACDs) 4-10
User and Group IDs 4-10
Can I ... ? 4-11
Programs 4-11
Commands 4-12
Starting Programs 4-13
Starting with the RUN command 4-13
Starting without the RUN command 4-14
Path restriction 4-15
Name restriction 4-16
Starting with the XEQ command 4-17

5. Where Does the Information Go?
To and From Your Terminal 5-2
What computers "know" 5-4
Understanding your command 5-5
Acting on your command 5-6
Destinations for Information 5-7
Computer memory 5-7
Random access memory (RAM) 5-7
Read only memory (ROM) 5-9
Disk—saving information 5-9
Tape—protecting information 5-10
Backing up files: full or partial 5-11
Printers 5-12
Spool Files/Print files 5-13
Output spool files 5-14
Input spool files 5-15

6. Behind the Scenes

Why Numbers? 6-2
Binary notation 6-4
ASCII code 6-6
Control and Escape 6-6
Letters and numbers 6-7
Files: ASCII or binary? 6-8
A Practical Problem 6-9
Addition—Computer Problem Solving 6-10

7. Commands

Types of Commands 7-2
Restricted commands 7-2
Console commands 7-2
Command Parameters 7-3
Optional parameters 7-4
Required parameters 7-6
Help with Commands 7-7
Syntax diagrams 7-8
Variables and keywords 7-8
Variable—required parameter 7-9
Variable—optional parameter 7-10
Keyword—not variable 7-11
The elements of syntax 7-12
Defaulted or Omitted Parameters 7-14
PRINT—A Closer Look 7-16

8. Command Files and Jobs

Command Files 8-2
Jobs and Job Files 8-2
Job or Session? 8-3
Starting jobs through HP Easytime/iX 8-5
Starting jobs through MPE/iX Commands 8-5
When You Run a Job 8-6
At the start 8-6
Job priority 8-7
Getting over the fence 8-8
Scheduling 8-8

Errors and your job listing 8-9

Index

Figures

1-1. The Parts on Your Desk 1-1
2-1. Files in Groups, Groups in Accounts 2-4
2-2. Files—In Groups, In Accounts 2-5
2-3. Hierarchical File System 2-6
4-1. Your Account Manager 4-5

Tables

3-1. Summary of MPE/iX CI Limits 3-5

1

What Is a Computer?

Unless it is your job to manage your HP 3000 Series 9X8LX computer, the answer that may come to mind is that a computer is whatever is sitting on your desk—the video terminal and keyboard that you work with every day. There is more, but these two pieces of *hardware* are your immediate connection to the computer.

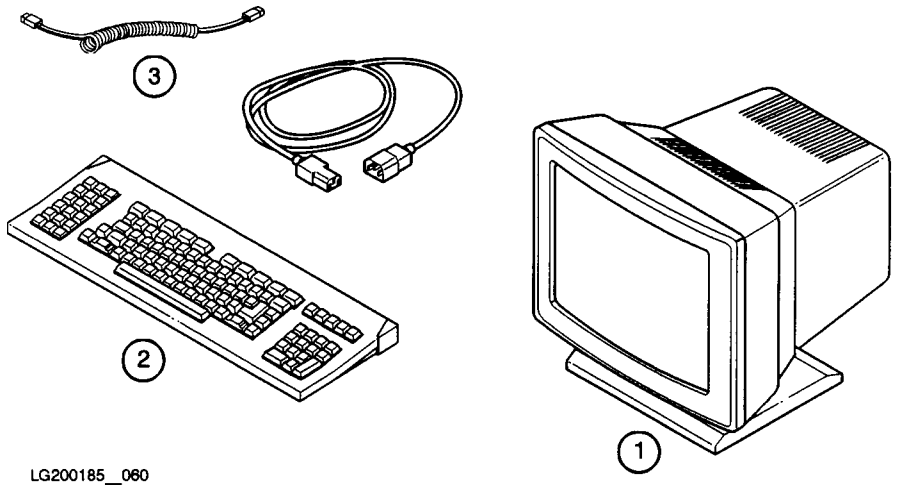


Figure 1-1. The Parts on Your Desk

1. video screen
2. keyboard
3. keyboard cable and power cord

Parts and Pieces—Seen and Unseen

Much of your computer is hidden in metal cases. Some of it is invisible to you. Behind the scenes are the fundamental parts of the computer. They consist of these elements:

hardware — All of the pieces that you can see, or touch, or pick up, or move around compose the *hardware* of your computer.

software — Recorded instructions that control the computer's operation. What you see of most programs is their names—you use their names to start them running—and their behavior, what they do when you use them.

Many of these recorded instructions are so well hidden that you may never be aware of their existence.

files — Recorded information. A typical file might be a letter that you write using the computer, or the accounting data that you enter into the computer.

Software is kept in files, too. There is a broad distinction between files: some files hold only information of one kind or another; other files (programs and their relatives) hold instructions that the computer can follow.

Hardware

Hardware consists of several different parts, each having its own role. At the minimum, your MPE/iX hardware consists of these parts:

- a system processing unit (hidden inside the computer cabinet)
- peripherals (some are hidden inside the computer cabinet)

system processing unit — The *SPU*. Often you will hear this called the *central processing unit* (*CPU*). This is the "brain" or "heart" of the computer. It is housed inside the computer box. Every instruction that you give to the computer passes through the electronic circuits of the SPU.

Turning on a personal computer sets in motion a number of automatic functions that prepare the computer for work—

usually in a matter of seconds or minutes. A multiuser system, such as your HP 3000 Series 9X8LX, requires time to start up, and it requires attention from someone who will perform many tasks to prepare the system for work.

In general, there are very few occasions for turning off your computer. An emergency of some sort—overheating or some other malfunction—requires turning off the power to your computer. Some maintenance tasks also warrant turning off the computer: a change in the computer's configuration (adding or removing certain vital hardware), or physically moving the computer to another location.

Peripheral devices

Peripherals are all of the other hardware connected in some way to the system processing unit. They are sometimes referred to as *peripheral devices*, meaning that they exist outside the system processing unit itself. Peripherals, such as an internal disk drive, might be built into your system and thus hidden from view, or they might exist outside the computer box, as printers and terminals do.

In one sense, a computer is a system processing unit plus everything else that might be connected to it.

terminal Your screen and your keyboard, together. With the keyboard, you send instructions to the computer. The screen displays your instructions as you type them and displays the computer's response. There may be one or many connected to your computer.

You may turn off your terminal at the end of the day, if you wish. Doing so will have no effect on the operation of the computer itself.

system console A special terminal that serves as the main terminal for your computer. The person who manages your system uses the console to control the day-to-day functioning of the computer. There is only one of these connected to your computer.

	This terminal is vital to the operation of the computer and should not be turned off, except for special maintenance of the system.
disk drive	Information that you create or record, and all the programs that the computer uses, are stored on disk drives. There may be one or many attached to your computer. Your MPE/iX computer comes to you with at least one disk drive built into the computer cabinet. You can add one more internal drive and other, external disk drives, too.
tape drive	Information (files and perhaps programs) that you do not frequently use are stored on tape. The tapes might be magnetic, much like the tape used in a conventional tape recorder, or they might be *digital data storage tapes* (*DDS*). Your HP 3000 Series 9X8LX computer comes with a DDS tape drive installed in the computer cabinet, but you can add other, external tape drives, too.
UPS	An external uninterruptible power system that provides up to 15 minutes of battery backup power to the computer system.
printer	Printers come in all shapes and sizes. Some are designed for extremely high speeds. Some produce very high quality, finished documents. If there is only one attached to your computer, it may be referred to as the *line printer*. There might be more than one, however.

Software

Software are the programs that control all of the actions of the computer. Software determines whether the computer acts as a text processor, graphic artist, accountant, mail deliverer, or the many other roles that computers can play.

The basic set of programs and files that come with your computer are called the *fundamental operating system* (*FOS*). You may, of course, add other programs from Hewlett-Packard or from other vendors who specialize in creating programs for the MPE line of computers.

Regardless of their source, three types of software will be of most concern to you.

operating system — This is the "master" program that oversees and directs all of the programs that are used on your system. The operating system in your computer is called MPE/iX, MultiProgramming Executive (MPE) with Integrated Posix (iX). iX in the name means that your operating system has more memory and more computing power than its predecessors.

command interpreter — The command interpreter (CI) is a special program whose primary purpose is to read what you enter at your terminal, determine whether you have entered a command that the computer can accept, and then determine what to do about it.

If you can log on, the CI is already at work and is waiting for you to issue a command. If the computer cannot carry out your command, the CI displays a message on your terminal. Such messages might consist of information about what is happening. They might be *error messages* that tell you that the computer cannot carry out your instructions. They might be *warning messages* to alert you that in some respect the action that you intended is not happening exactly as you expected it to occur.

other programs — The number and variety of programs that could be in your computer, waiting for you to use them, may be surprising, and more are being created every year.

The programs most commonly found on MPE/iX computers perform tasks such as these:

Parts and Pieces

electronic mail	sending and receiving messages
word processing	creating and printing documents of all sorts (sometimes called a *text editor*)
data processing	organizing and managing immense volumes of information, such as inventory and orders management, mailing lists, and the like (usually recorded in a *database*)
forms creation	mimicking the kinds of paper forms that businesses use to collect, assemble, and organize information (frequently used in conjunction with a data processing program)
accounting	accounts payable and receivable, general ledger, payroll, billing, and many others
process control	controlling other machinery in the manufacture of products, as well as monitoring and analyzing the performance of other (electronic) machinery

This list is far from exhaustive.

HP Easytime/iX is a program, too—one that is designed expressly for managing your MPE/iX computer and using its many functions. You will find detailed information about using HP Easytime/iX in *Using Your System*.

Programs are sometimes called *applications*. And, some programs on the MPE/iX operating system are referred to as *subystems*. The distinction between programs, applications, and subsystems is of interest to the people who design them. In everyday use, the terms are interchangeable and mean the same thing.

Files

Files—recorded information or instructions—fall into two broad categories, *data files* and *executable files*.

data files	Data files include the letter or document that you might create, the database holding immense quantities of information, personnel records, financial records, and many others. What they have in common is that the information contained in these files is created and recorded by programs of one kind or another.

executable files include all of the programs—recorded instructions—that direct the computer.

Command files and *job files* are text (data) files. Unlike most data files, command files and job files contain instructions that the computer can interpret and carry out. Superficially, however, they behave like executable files. You will find more information about command files and job files in Chapters 8 and 9 of this book, and in the book *Task Reference - HP 3000 Series 9X8LX* (B3813-90009).

What Are Programs?

> A computer without a program is useless, as is an orchestra without music to play.

Programs are what programs do. In fact, unless you use a program to *do* something, it is just another file taking up space on your computer disk. *What* any program does is to carry out instructions. Precisely *what* any particular program does depends upon the instructions that have been designed into it.

Programs accomplish what they do by doing such things as these:

- acting as a messenger, translator, and negotiator between you and the operating system and the system processing unit
- solving one particular problem—or a group of related problems—in ways that save you time and effort
- taking information from somewhere
- displaying information
- saving or storing information
- sending information from one place to another
- recalling (remember) information
- following rules
- carrying out instructions
- manipulating information to produce new results

In general, programs (such as HP Easytime/iX) relieve you of the need to understand the workings of the operating system. They take information from you and do some of the needed processing. They make direct contact with the operating system to move your information where you want it, in the form you want it. In this sense, programs are your agents. Acting on your behalf, they draw up and execute "contracts" for the services provided by the operating system and the system processing unit.

The most sophisticated programs allow you to solve a problem, or to accomplish a task, in ways that are already familiar—or at least in ways that are less mysterious than the workings of the operating system.

A spreadsheet mimics the appearance and function of an accounting ledger. You may enter numbers (and words) in positions that define their relation to each other. You may define one number as being the sum (or difference, or product) of other numbers. If you later change one of the component numbers, the number you defined as being the sum (or difference, or product) changes with it automatically, almost instantly.

The most sophisticated programs do something else, as well. They protect you from—at the very least, they alert you to—impending problems. Using a spreadsheet in which you have entered numbers and words, it is surprisingly easy to tell the program to add a number and a word. In a moment of haste, you might tell the program to give you the result of 236.5+"January Sales". The system processing unit cannot perform this operation, and it would interpret the attempt as an error. An "alert" spreadsheet will recognize the problem and give you some message pointing out the problem and perhaps even suggest a resolution.

The rules for using any program are unique to that program. The documentation that comes with a program is your best source of information. Some programs actually suggest how to use them; some provide tutorials to help you learn how to use them; others provide helpful information that you can display on your terminal by pressing one or more keys.

Whenever possible, draw on the experience of others who have used a particular program.

How Do Programs Do What They Do?

If you have ever followed a recipe to cook a meal, or assembled something using printed instructions, you have gone step by step through a sequence of instructions, much the way a computer goes step by step through a program.

Mama's Nice, Simple Tomato Sauce

Ingredients

- fresh tomatoes, 3 or 4, cut up (peeled and seeded, if you wish). Save the juice.
- olive oil, one tablespoon
- garlic, one clove, minced
- onion, one-half, diced
- fresh basil, two or three leaves
- salt and pepper to taste

Directions

1. Warm the oil in a heavy pan or skillet.
2. Add garlic and onion, cook gently until onions are translucent.
3. Add tomatoes, tomato juice, and basil.
4. Add salt and pepper to your liking.
5. Simmer and allow the liquid to reduce.
6. Taste and correct the seasonings.
7. Serve over hot, cooked pasta.

Everything you need to know in order to prepare this sauce is given in this recipe: what you need to buy, what you need to do. Computer programs do for computers what recipes do for cooks—organize information and provide instructions for carrying out tasks.

Sometimes these instructions are no more complex than following a sequence of unchanging steps. More complicated methods require analysis and decision.

> "If the cost of production exceeds revenues by 2 percent in any three consecutive months.... "
>
> "While lemons are on sale, buy lemons and make pies.... "
>
> "Buy lemons and make pies until the price of lemons has increased by 6 percent ... "

The Idea Behind Computing

In some ways, the operation of a computer mimics the kind of thinking that you do every day. When you set out to solve a problem or accomplish a task, you are likely to have a method, a way of doing things, that works for you. Many of your methods are so automatic that you do not think about them. Many of the computer's methods seem automatic; however, *everything* that the computer does—from the simplest to the most complex task—is controlled by strict and specific instructions (programs).

If you have ever written down a list of numbers and added them up, you have used procedures—*input*, *processing*, and *output*—that parallel the operation of a computer.

A problem in arithmetic

```
  234  What is your method for adding
   19  these numbers?  Do you first
    8  add all the numbers in the right
  611  column, or do you add 234 to 19
  ---- and then add 8 to that sum, and
  872  so on...?
```

Any method that consistently yields the correct solution is a good one.

Where the numbers come from, what you do with them, and where the results go are important—to you and to the computer.

- The numbers come from somewhere. Perhaps they come from the records of your monthly spending.

  For the computer, information that comes from somewhere is called *input*.

- You do something with the numbers: you add them to produce a sum. The sum did not exist until you performed the addition.

  Using existing information to produce new information (or to change the content or nature of existing information) is called *processing*.

- The result (the sum) you obtain goes somewhere. You might add it to the records of your yearly spending.

  Sending (or saving or storing) information somewhere is called *output*.

Your mental computer

Over and over again throughout this process of adding numbers, you are taking in information, recording information, moving information from one place to another, and manipulating existing information to produce new information.

In our day-to-day lives, we use input and output so automatically that we have to stop and think before we realize exactly how much input and output is occurring.

We you see a number written on a piece of paper, the image of the number enters your mind. That is input. If you then turn around and write that number on another piece of paper, you move the image in your mind onto the paper. That is output.

It does stretch the imagination, but what the paper "receives" when you write down a number is input for the paper.

If you write down a series of numbers on the paper and then perform addition, you are engaged in a startling number of input-output exchanges.

Somewhere between taking two numbers into your mind and writing down their sum, the mental steps required to perform addition come into play. In these steps, your mind does what the system processing unit of a computer would do: joins two (or more) numbers to create a third number. This is *processing*.

> Input and output are so closely associated in the operation of computers that they are almost considered a single process: *Input/Output*, sometimes abbreviated as *I/O*.

Because the sum is important in your budget, you "output" it once again, perhaps to an accounting ledger, and record it for future use. Having recorded this important information, you are free to turn to other tasks, assured that you can retrieve the information when you need it. The computer, too, records

information (in files—in its own memory, on disks, or on tape) for later retrieval.

> For the computer, "forgetting" something happens when you turn off the power or erase a file from a disk or a tape.

So a computer is....

You can think of a computer as the sum of all of its pieces of hardware. If you include all of its software in that definition, you are closer to the truth. And if you go no further, you have an entirely workable definition.

Here you are invited to consider the computer, too, as the sum of the processes (actions) that it performs.

Considered as the sum of its processes, a computer truly is whatever the computer is doing when you use it. This may sound strange. But consider what a carpenter is. A carpenter is someone who.... Now, list all the things that a carpenter does. The list could be very long.

The carpenter may also be a parent, a loving spouse, an avid photographer, an active participant in community affairs. The human being who is all of these things is doubtless a very complex individual. Yet, when you hire the carpenter to build something, you hire the knowledge, skills, and activities that make the person a carpenter. In that sense, a carpenter *is* what a carpenter does. In that sense—in a very restricted sense—a carpenter is a "process."

In a similar fashion, the computer follows command or program instructions in order to take on the role of report writer, document printer, accountant, record keeper, assembly line coordinator, product designer, mail deliverer.... The list goes on and on.

In its simplest activities, a computer takes and records information.

When necessary, it "remembers" what it recorded. When instructed to act, it applies appropriate rules (found in programs) to the information it has and displays or records the results. When necessary, it moves information from one place to another.

The information might be numbers, or letters of the alphabet, or words, or illustrations, or some combination of them. Some computers even work with sounds.

What is your computer right now? It is what you instruct it to do—assuming that it is able to carry out your instruction.

What computers do

- take information from somewhere (input)
- transfer information somewhere else (output)
- display information (output)
- record information (output)
- recall (remember) information
- follow rules (programs or commands)
- carry out instructions (programs or commands)
- manipulate information to produce new results (processing)

Interruptions

Now and then you may find that your work on the computer is interrupted, usually when the person managing your computer has vital administrative or maintenance duties to perform.

Shutdowns

There may be times when the person managing your system warns you of an impending *shutdown*.

A shutdown is not the same as turning off the power. Instead, it involves stopping the operating system (reducing the computer's operations to a bare minimum), in order to perform certain kinds of maintenance. This will not happen often.

Backups

From time to time, the person who manages your system needs to *back up* the files that all of the users have created. Backing up files is a method of copying them to a tape, so that they will be protected from loss.

Backups are vital and should be performed at regular intervals in order to ensure that no vital data is ever lost.

System too busy

As your system grows, as more and more people come to use the computer, there may come a day when it appears that the computer is running more slowly than usual. As fast as computers are, they can become too busy to do everything with efficiency.

The person managing your system may decide to reduce the computer's work load temporarily, letting some work continue and putting other work on hold.

What to do

In any of these circumstances, the person managing your system should warn you by sending a message to your video screen. You should receive these warnings in time to take appropriate action.

Usually you will be asked to save the work that you are doing and log off until the maintenance or administrative work is completed, or until the load on the computer has decreased. Later, you will be notified that you may log on to the computer again and resume your work.

Do be sure to wind up the work that you are doing and save the work that you have completed so far. When you resume your work, you will find that you can pick up again where you left off.

2

Where Am I? Logging On

Personal computers are designed for one person to use. Once you have turned on a personal computer, you are ready to do something.

Your MPE/iX computer, however, is designed for many persons to use, and on any given day, all of them might be using the computer at the same time. Without some method of keeping order, it could become confusing.

Maintaining Order

Your MPE/iX computer maintains order on the system through two closely related forms of organization.

logging on

Logging on is your way of fitting yourself into the computer's organization. By logging on with the `HELLO` command, you answer the computer's most important questions.

- Who are you?
- Where do you want to fit into my organization?
- Do you have the authority to do this?

Logging on identifies you to the computer. Just as important, it tells the computer whether you have the authority to do certain kinds of work on the computer.

the account structure

Would you take every single piece of paper related to your business and throw them all into one box? You might, if you have a *very* small business. More likely, you want to organize all your papers into some structure that allows you to find things easily.

For many businesses, the filing cabinet with drawers and folders provides a place to put papers. More important, it provides a way to keep together those pieces of paper that belong together and to keep separate those pieces of paper that are unrelated to each other.

The computer's account structure provides the same solution for those who use the computer.

- An *account* is like a filing cabinet.

  Unlike the filing cabinet in your office, an account may be as large, or as small, as you need. You may have almost as many accounts on your computer as you like. Accounts do take up room on the computer: in its memory and on its disks. But if you are willing to buy more and more disks, you can create accounts almost to your heart's content.

- *Users* are somewhat like the keys to a filing cabinet.

  You might create an account called `XM661A`. And you might specify that a user called `M661` is entitled to *access* (open and use) the account that you created. It might make more sense to call an account `MYACCT`. It would certainly be easier to remember. And it might make more sense to call a user `JOHN`. That, too, is easier to remember. The choice is up to you—more accurately, the choice is up to the person who sets up and manages your computer system.

- A *group* is like one of the drawers in a filing cabinet.

  Unlike the drawer of your filing cabinet, a group may be as large, or as small, as you need. And, within the limits of your computer's disk space and memory limits, you may have as many groups in an account as you need.

- *Files* are comparable to the pieces of paper (or folders) that you store in a drawer of the filing cabinet. In the computer, files are kept in groups. Again, within the limits of your computer's disk space and memory limits, you may have as many files in any group as you need.

- *the hierarchical file system*

  The MPE/iX file system has been expanded so that it is *hierarchical* (tree structured) and can contain files at many different levels. This organization provides a special kind of file called a **directory**. Instead of holding data, directories contain lists of files and pointers to those files. A directory can also contain other directories. For more information on the *hierarchical file system*, refer to the book, *New Features of MPE/iX: Using the Hierarchical File System* (32650-90351).

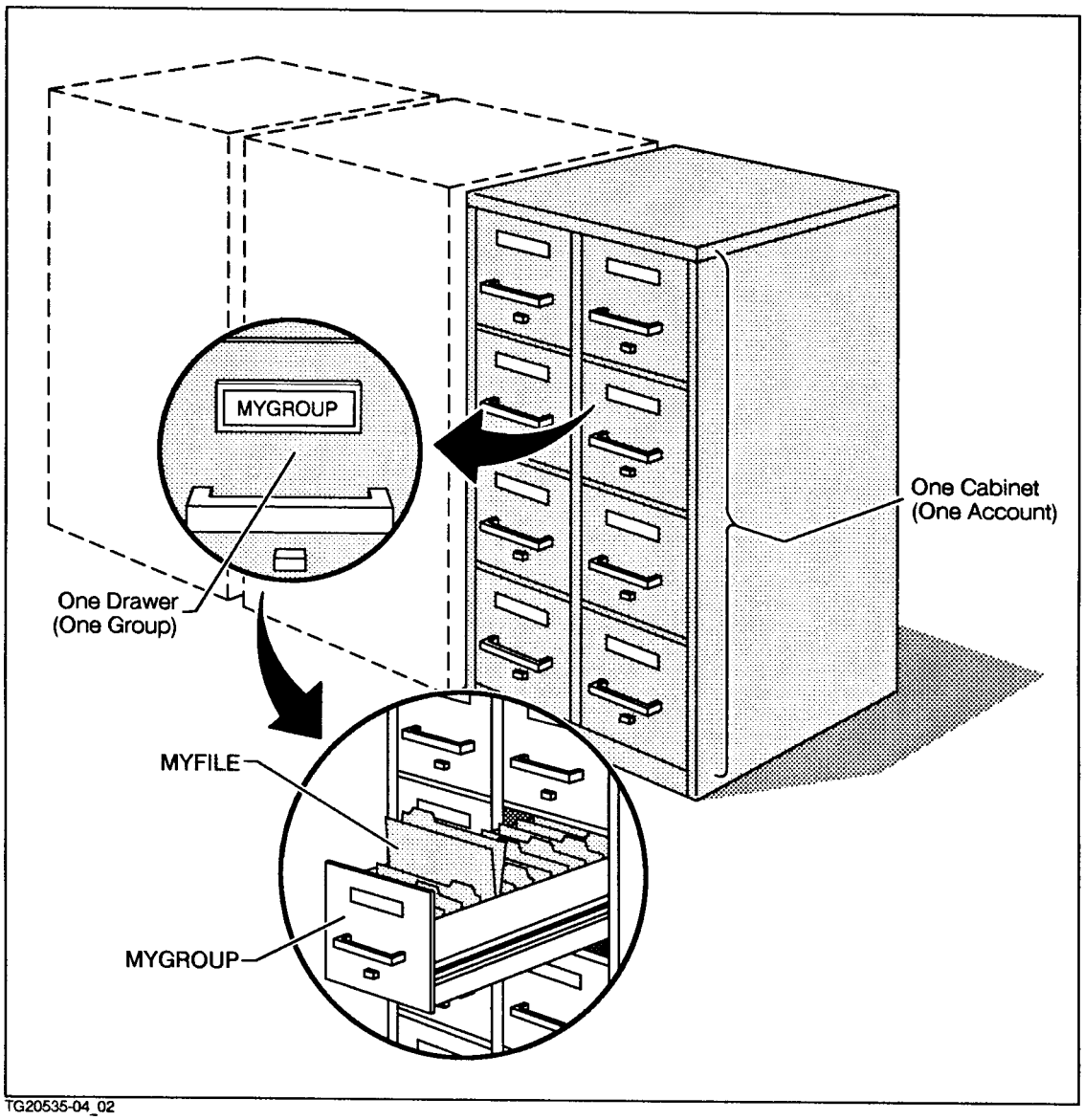


Figure 2-1. Files in Groups, Groups in Accounts

Just as important, the account structure offers a method of protecting sensitive files from tampering, and sensitive programs from abuse.

You may create passwords that anyone wishing to use the system must know before he or she can log on.

- Accounts may have passwords.
- Groups may have passwords.
- Users may have passwords.

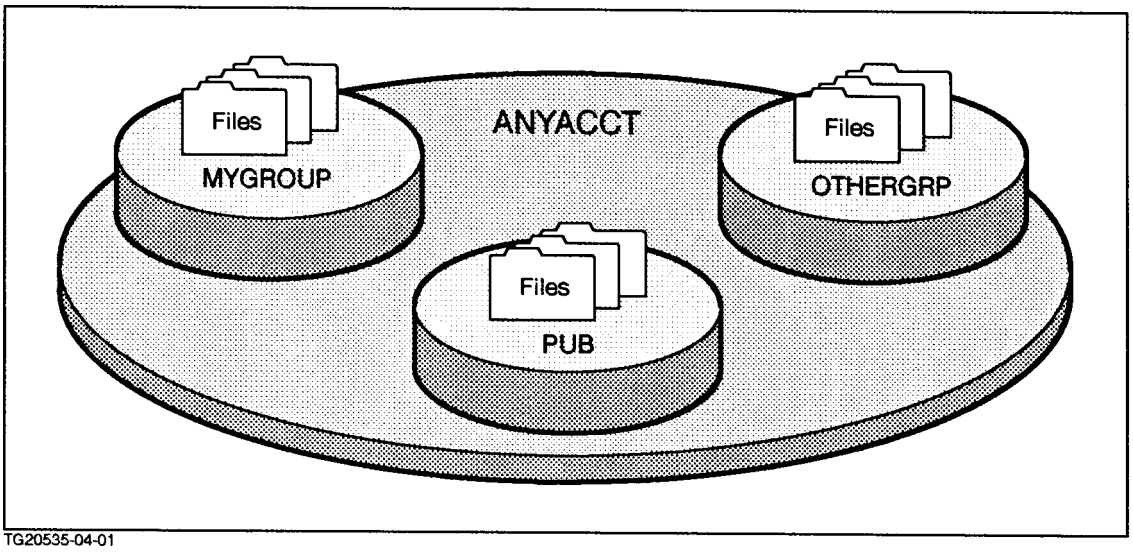
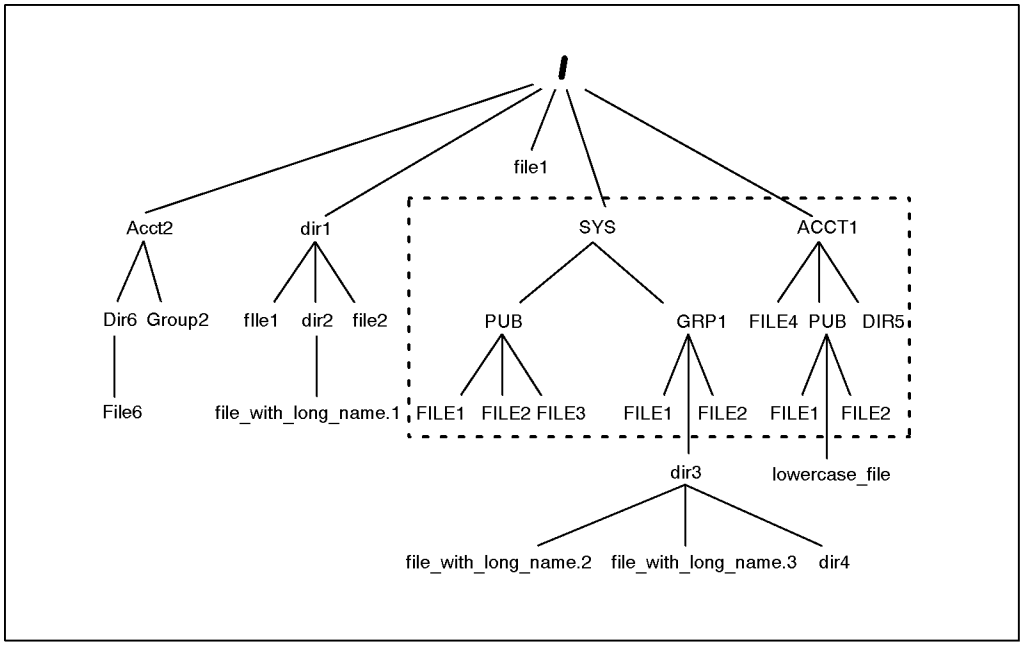


Figure 2-2. Files—In Groups, In Accounts

Maintaining Order

In addition to the traditional group, file and account structure mentioned previously, you can now use the *hierarchical file system* shown below. The *hierarchical file system* is tree structured and can contain files at many different levels. You can also create files and directories under accounts. For more information on the *hierarchical file system*, refer to the book, *New Features of MPE/iX: Using the Hierarchical File System* (32650-90351).



LG200208_009

Figure 2-3. Hierarchical File System

Gaining Admission

When you want to work on a file in one of your filing cabinets, you may simply go to the right filing cabinet, open the right drawer, and pull out the file folder that you need.

Working with the computer, however, you must specify *which* filing cabinet and *which* drawer you want to open.

The computer insists that you provide the answers to its questions.

- Which user are you?

  Do you know the user password (if there is one)?

- Which account do you want to use?

  Do you know the account password (if there is one)?

- Which group (in this account) do you want to use?

  Do you know the group password (if there is one)?

Logging on is your way of answering these questions. Only after you have provided the correct answers does the computer permit you to use the file that you need.

Notice that the computer does not care which file you want to use. It simply opens the right cabinet (account) and the right drawer (group)—provided you have correctly answered all of the computer's questions.

Starting a Session

Logging on with the HELLO command starts a computer *session*—a dialog between you and the computer—and permits you to give instructions at your terminal, one at a time, to the computer. The computer attempts to carry out each instruction as soon as it can.

If you enter something that the computer does not understand, or that it cannot for some reason execute, the computer will inform you by displaying an *error message* on your video terminal. Receiving an error message does not mean that you were wrong. You may have intended to enter exactly what you entered. It does mean that the computer has found something about your entry that does not fit the rules that it must obey.

This is a simple log on:

```
HELLO JOHN.ANYACCT,MYGROUP
```
Return

HELLO is the command to log on and start a session.

Here the user JOHN is starting a session and opening the group called MYGROUP in the account called ANYACCT.

Notice the order in which the log on is specified.

HELLO	JOHN	.ANYACCT	,MYGROUP
command	*user name*	*account name*	*group name*

> This order is vital, and so is the punctuation (the period before ANYACCT and the comma before MYGROUP). Computers are strict about order and about punctuation.

When you log on

If passwords are required, the computer will ask (*prompt* you) for each of them, one at a time, and wait for your response. In that case, you would have to enter each password correctly in response to the computer's request.

The passwords that you type will be invisible to you and to everyone else. The computer will read what you enter, but it will not display the passwords on your video terminal as you type them. This prevents others from looking over your shoulder to discover the passwords that you are using.

> The computer limits you to three consecutive attempts at logging on with any one logon identity (*user.account,group*) or three attempts at any one password. After three consecutive failures at any one logon identity or at any one password, the computer will close its communication with you—another security provision designed to keep unauthorized persons from using the computer.
>
> You must then reestablish communication with the computer in order to try again. How you establish this communication depends upon the electronic communications method established when your computer is set up at your site.

I have logged on—now where am i?

This is one way of logging on:

```
HELLO JOHN.ANYACCT,MYGROUP
```
Return

Logging on this way opens one "drawer" of a computer "filing cabinet."

- The filing cabinet that you opened is called `ANYACCT`.
- The specific drawer that you opened is called `MYGROUP`.

Here is another way of logging on:

```
HELLO JOHN.ANYACCT
```
Return

Logging on this way opens still another drawer—a rather special drawer—in the filing cabinet called ANYACCT. But that is a subject for "Logging on Without a Group".

But where are the filing cabinets in the computer?

The address of files

Accounts (the filing cabinets), their groups, and the files within those groups are kept on the computer's disk or disks, which correspond to file rooms in an office.

The log on does not by itself give you direct access to any particular file. Rather, it specifies a group in an account where the file is to be found.

If a file called MYREPORT is found in the MYGROUP drawer of the ANYACCT filing cabinet, its full address (location) is this:

```
MYREPORT.MYGROUP.ANYACCT
```

MYREPORT	MYGROUP	ANYACCT
file name	*group name*	*account name*
file	drawer	cabinet

The full names of files

It happens—and not at all by accident—that the complete address (location) of a file in the *account structure* is exactly the same as its full name.

```
MYREPORT.MYGROUP.ANYACCT
```

> When a file name is shown in this fashion, it is called a *fully qualified file name*, and it is the full and correct address of the file. For the computer, this is the equivalent of saying:
>
> - the file `MYREPORT` ... (file)
> - in the group `MYGROUP` (group) ...
> - in the filing cabinet called `ANYACCT` (account)

There might be many files in the same group of the same account. These three files are found in the same group and account in the account structure:

```
ACCTRPT.MYGROUP.ANYACCT
BUDGRPT.MYGROUP.ANYACCT
TAXRPT.MYGROUP.ANYACCT
```

Each has the same group name and account name. But each has a unique, fully qualified file name. Each has a unique address or location in the computer. It must be so—two files cannot have the same fully qualified file name on the same disk.

When would you use the fully qualified name (address) of a file? You would use it whenever you need access to that file but you are not logged on in the right group or account.

> Access to a file is governed by the security provisions in effect on your computer. While you may be able to find a file, you might—or might not—be able to see it, copy it, use it, or change it.

Where is that file?

The day will come when you want to find a file and you cannot remember where it is. Unlike the filing cabinet in the corner of your office, the computer can tell you where to find a file, provided the file exists.

If you remember the first name of the file, the computer can find it quickly. If you remember only a part of the name, the computer can still find it—or at least show you file names that are close to the one that you are searching for.

The `LISTFILE` command does the searching.

If you did not know where to find the `TAXRPT` program, you would use `LISTFILE TAXRPT.@.@,6` to locate it. If `TAXRPT` is found on the system, the `LISTFILE` command will tell you which group and which account contain `TAXRPT`.

- `@` is a *wild card* character, an all-purpose name hunter. To the computer, `@` means "any combination of characters, numbers, or letter—as few as none, as many as eight."

  `LISTFILE TAXRPT.@.@` tells the computer: "Look for the `TAXRPT` file. Look in every group. Look in every account."

- `6` is a code for the `LISTFILE` command. Its meaning is "display the fully qualified file name(s) (*file name.group name.account name*) of any file(s) you find."

  This command could also be entered as

  `LISTFILE TAXRPT.@.@,QUALIFY`

  `QUALIFY` has the same meaning as `6` for `LISTFILE`.

You might, however, have many file names that begin with the letters `T A X R P T`—`TAXRPT91, TAXRPT92, TAXRPT93 ...` . And, you might recall that all of them are in the `ANYACCT` account.

If that were the case, you could enter this:

```
LISTFILE TAXRPT@.@.ANYACCT,QUALIFY
```

This tells the computer: "Look for any file name beginning with `TAXRPT`, in any group, in the `ANYACCT` account."

Suppose, however, that the files were named this way: JNTAXRPT, FBTAXRPT, MRTAXRPT, MYTAXRPT ... .

In that case, enter this:

```
LISTFILE @TAXRPT.@.ANYACCT,6
```

"Look for any file names that *end* in TAXRPT in any group in the ANYACCT account."

You might, as an experiment, enter this command:

```
LISTFILE @.@.@,QUALIFY
```

This will find—and display—the fully qualified names of *all* of the files on your system. The list could be quite long, and the display could take time. Pressing Break will *break* the command and its display and return the prompt to your screen.

> Many commands—but not all of them—respond to pressing the Break key. Pressing Break terminates some commands and merely suspends others. You may use the ABORT command to terminate the suspended command. You may use the RESUME command to resume the operation of a suspended command.

LISTFILE options

With this release, LISTFILE contains additional ways to display information about files. For more information and examples, refer to the book *Commands Reference - HP 3000 Series 9X8LX* (B3813-90011). User names, such as JOHN, have no bearing on the location of a file, a group, or an account. They are, however, *indispensable* for logging on to an account.

Equally important, when you combine a user name with an account name (JOHN.ANYACCT), you identify the *creator* and "owner" of a file.

Whoever logs on using JOHN.ANYACCT and creates a file—in any group in the ANYACCT account—is identified as the creator and owner of that new file.

> You, as a person, are not the owner of this new file. The owner of this newly created file is JOHN.ANYACCT—the log on identity used when the file is created.

Someone logging on with ALICIA.ANYACCT might be unable to use the newly created file—*unless* the owner identified as JOHN.ANYACCT uses the RELEASE command to suspend the normal security on that file. Security is restored to that file only after JOHN.ANYACCT uses the SECURE command.

Ownership information is attached to the computer's record of any file that JOHN.ANYACCT creates. ALICIA.ANYACCT might create files, too, and the computer would know that certain files "belong" to ALICIA.ANYACCT and that others "belong" to JOHN.ANYACCT.

User names have still another purpose that is described in "Logging on without a group" and in "Capabilities" in Chapter 4.

Session names

Still another kind of name might be found in a log on. In this example, the word BUDGET has been added:

```
HELLO BUDGET,JOHN.ANYACCT,MYGROUP(Return)
```

BUDGET is a *session name*.

A session name has no bearing on the location of a file, group, or account. It has no bearing on the user name JOHN, either. So what is its value?

It is entirely possible to find that two or more people have logged on to the same group, in the same account, using the same user name (HELLO JOHN.ANYACCT,MYGROUP(Return)).

If more than one person has logged on to a group using the same user name, each might add a different session name to his or her log on to help minimize confusion. In most cases, your printer will add any session name to the banner page that precedes each print out (as well as the log on identity of the user doing the printing).

Session names are a convenience for the *people* using the computer. The computer will not become confused if two different people log on using the same user name, the same account name, and the same group name, even if those people neglect to use a session name. Every time someone logs on to the computer, the computer starts a new session and assigns that session a unique *session number* (such as `#S373`). You will see session numbers if you use the `SHOWJOB` command.

You may choose any session name that you like when you log on, provided the name begins with a letter is followed by no more than seven other letters or numbers, or a combination of letters and numbers.

> There is no requirement to include a session name when you log on.

Password protection

Ownership is one form of protection for files. Using passwords is another form of protection.

Passwords may be attached to the following parts of a logon:

- a user name
- an account name
- a group name

Logon Element	Account Structure Element	Possible Password (examples)
`BUDGET`	session name	none possible
`JOHN`	user name	`ACACIA`
`ANYACCT`	account name	`FIN91`
`MYGROUP`	group name	`RPT91`

Like other names used to identify files or to log on, a password may consist of as few as one or as many as eight characters. The first character must be a

letter of the alphabet. Any remaining characters may be letters or numbers or a combination of them.

The person in charge of your computer system may change or remove account passwords. The person in charge of an account, may change or remove group passwords. Unless you are one of these people, you cannot remove or change the passwords attached to an account or to a group.

You may, however, use the PASSWORD command to change the password attached to your own user name.

Logging on without a group

This log on puts you in the MYGROUP group of the ANYACCT account:

```
HELLO JOHN.ANYACCT,MYGROUP
```
Return

This log on puts you in *some* group of the ANYACCT account:

```
HELLO JOHN.ANYACCT
```
Return

But in which group do you find yourself?

Where that log on takes you depends upon two factors:

- which user name (of those attached to the account) you use
- which group is assigned to that user as a *home group*

Recall that user names are attached to accounts. In turn, a home group may be attached to a user name.

A home group may be any existing group within the account. When a home group is attached to a user name, that group becomes the *default* log on group for that user name.

HELLO JOHN.ANYACCT Return is an example of a default log on. The user JOHN did not specify the name of a group. As a result, this log on leads into the ANYACCT account and—by default—into the home group for the user JOHN.

If the home group for user JOHN is the MTKG group, then this log on

```
HELLO JOHN.ANYACCT
```

is equivalent to—and has the same effect as—this log on:

```
HELLO JOHN.ANYACCT,MKTG
```

If no home group has been assigned to `JOHN`, then `HELLO JOHN.ANYACCT` will fail.

Shared group

It is fairly common to designate one group within each account that is open to all users who log on to that account. This "shared" group is open to anyone who can log on to the account.

A shared group provides a "place" for any programs and files that deserve to be available to all of the users of an account. A shared group belongs to all of the users of that account.

Those users do not need to specify the password for the shared group when they log on to that group. Of course, they may still need to be able to enter the passwords for the user name and the account name.

> One common convention is to allow `PUB` to be the shared group. `PUB` is an abbreviation for the English word "public," as in "a public park," a park open to everyone. Every account that is created on your system automatically has a `PUB` group.
>
> It is not unusual to make the *shared* group the *home* group for one or more users of an account.

SYS—A Special Account

Your HP 3000 Series 9X8LX computer arrives with one special account already in existence, the SYS account.

SYS is the account from which the computer is first started and from which it is managed and controlled day by day. Consequently, access to it should be tightly restricted by exercising careful control over the passwords attached to SYS and to its assigned users, especially to the users identified as MANAGER and OPERATOR.

Two users (logons) are of special importance to the operation of the computer. One is MANAGER.SYS. The other is OPERATOR.SYS.

MANAGER.SYS

The user MANAGER has the highest level of authority on the system. The user identified as MANAGER is responsible for starting and setting up the computer when it arrives in your office.

Only this user has the authority to install the computer and to set limits on the internal computer resources available to other users of the system.

OPERATOR.SYS

The user OPERATOR has authorities that are almost as extensive as MANAGER.

It is from the SYS account that the operator controls the day to day operation of the computer, seeing to the needs of those who are using the system.

The PUB group

Only MANAGER, OPERATOR, and specially designated users can log on to the SYS account. But the files and programs located in the PUB group of the SYS account are available to any user of the system, no matter how you log on.

This is the ideal group in which to put those files and programs that everyone will (or might) use. It is not the place for files or programs that deserve to be restricted in their use.

The account manager

Each account that is created must have one special user, the account manager. This user has the highest set of authorities within the account, including the authority to create new groups and new users and to assign passwords to groups and users.

3

What Are Files?

Programs take, send, record, and retrieve information. From the perspective of the computer, files are the only source and the only destination for all of the information that the programs manage for you.

If you have kept records of any kind for any purpose, you have worked with files. Keeping related pieces of information together, in one place, is a loose but workable definition of a file.

Computers, however, go beyond this loose definition of a file. For the computer, a file is the following:

- a collection of related pieces of information kept in a consistent form.
- a source from which information is retrieved (a computer "reads" a file).
- a destination to which information is sent (a computer "writes" to a file).

Where Are Files?

Files are kept on disks (unless they are stored on tape), and it is logical to assume that computers keep the contents of a single disk file in one, single location. In fact, computers almost never do that.

What they really do is to keep the contents of a single disk file in one identity. You control the identity of files by giving them names.

Disk memory

A phonograph disk is the device that comes closest to the behavior of a computer's memory disk. But the phonograph disk allows you to play back (retrieve) only what is recorded on it. Magnetic or digital tape of the sort used to record music does permit both recording (input) and playback (output).

Still, tape has one serious drawback. In order to hear the music recorded in the middle of the tape, you must first play the music at the front of the tape and wait. With a phonograph, you can position the tone arm of the phonograph to play music from any place on the phonograph record.

Computer memory disks combine the best of both: like tapes, disks can record and retrieve information. Like the phonograph, computer disks have a "tone arm," called a *read-write head*, that records and retrieves information from any location on the disk.

There is one serious complication. At any moment, several people may be recording or retrieving information. None of them wants to wait his or her turn while someone else records or retrieves information.

Two solutions are combined in computer disk management:

- The recording and retrieving of information occur at breathtaking speed.
- Information is recorded in small chunks, called *segments*, that are scattered in locations all across the disk.

This sounds like a prescription for chaos. In fact, it is accomplished in a highly efficient manner. More to the point, you do not need to worry that the computer will lose one of the segments of any of your files.

File recording

File segments are *written* (recorded) in *tracks* on a disk. A track, like the groove in a record, is a prescribed path for the read-write head of the disk to follow.

If three people are recording information at the same time, the computer *writes* a segment of information for the first person, a segment for the next person, and a segment for the next person. Then it repeats the process beginning again with the first person. It continues in this fashion until all of the information for all three people is written to the disk. Along the way, there is no attempt to keep all of the related segments together. Instead, the computer creates a map for itself to direct it to all of all the segments it records.

> While everyone using the computer is recording and retrieving information from the disk, the computer itself is busy reading and writing information for its own purposes, keeping track of everything it needs to know about its own operations and about your work.

Pointing to the right pieces

Controlling the location of a file on the disk is the computer's job. Your part consists of controlling the identity of the files by giving names to the files you create. For the most part, you will give names to files while using a particular program that stores information in files. Using the EDIT/3000 program's command `KEEP` allows you to name the file that you save to disk. In a few cases, however, the program itself will give names to files that it uses for its own purposes.

> Many programs create *temporary files*. These serve as holding places—"scratch pads," if you prefer—while work is in progress. When the work is completed, the program that created the temporary file writes the final version of the work in a *permanent file* and erases the temporary file.

The computer keeps a record of the names (and creators) of all files created. This record is kept in still another, special file that the computer itself creates and names.

Along with the name of each file, the computer records the location on the disk where the first segment of every file is to be found. While you are recording information in files, while the computer is laying down segments of files, the computer keeps careful records of the location of each segment of every file.

The naming of files

Every computer system has rules about the names that you give to files.

Within those rules, you are free to have user names, account names, group names, and passwords of almost any sort. For a summary of MPE and HFS syntax and naming rules, see the following Table.

Table 3-1. Summary of MPE/iX CI Limits

Feature	HFS Syntax	MPE Syntax
Maximum directory depth	512 (`/1/2/ ... /512`)	3 (file, group, account)
Maximum number of characters in a component	Up to 255 for files or directories under HFS directories	8 for accounts, groups, or files
Use of MPE syntax or HFS syntax	Initial slash (/) or dot (.) in the CI means use HFS syntax; only HFS syntax is used in the MPE/iX shell	Lack of an initial / or . in the CI means use MPE syntax
File referencing direction	Top-down (`/ACCT/GRP/file`)	Bottom-up (`FILE.GRP.ACCT`)
Pathname separators	Slashes (/)	Dots (.)
Case sensitivity	Yes (`FILE1` and `file1` are two different files.)	No (Lowercase automatically upshifts to uppercase.)

Table 3-1. Summary of MPE/iX CI Limits (continued)

Feature	HFS Syntax	MPE Syntax
File location	Under any directory, /, account, or group	Under a group only
Maximum characters in pathname	253 for relative pathnames (**./253chars**), and 254 for absolute names (**/254chars**)	26 (8 times 3) +2
Lockwords	Not allowed	`MEMO/A3` is a file named `MEMO` with a lockword called `A3`
Specifying remote environment	Not allowed	Remote environment specified using *envid*
File equations	(1) Allowed only on right side of equation; (2) Cannot backreference a file using HFS syntax	Allowed on both sides of the equation; backreferencing done by preceding name with asterisk (*)

Table 3-1 summarizes the primary differences between HFS and MPE syntax as it operates in the command interpreter (CI). The CI poses some constraints on component and pathname lengths, and maximum directory depth due to the length of the command buffer. (The term *component* refers to a file, account, group, directory, or fileset).

Unless it is your job to set up the computer, you will probably find that whoever did set up the computer has already established account names, group names, and user names by the time you start working with the computer.

The Location of Files

The *account structure* of the computer is analogous to a room full of filing cabinets—a place, or places, for the orderly storage of information in files (Figure 2-1).

This picture is not literally accurate. From a technical point of view, it is a little misleading to think of computer files as being in places such as the drawer of a filing cabinet. But that description helps to explain the organization of the account structure. If the image of a cabinet full of drawers and files is useful to you, then keep it in mind.

If you are looking for a file called `MYREPORT` and you know that it is in the filing cabinet called `ANYACCT`, in the filing drawer called `MYGROUP`, you would log on in this way:

```
HELLO JOHN.ANYACCT,MYGROUP
```
Return

> Notice that although you are looking for a particular file, it is not necessary—it is not even possible—to name the file when you log on. You need to name the filing cabinet (account) and the drawer (group) and the key (user name) attached to the filing cabinet.

If you had logged on to a group called `YRENDSUM` in the `ANYACCT` account and you wanted to see the contents of the file `TAXRPT` in the `MYGROUP` group, you would do this:

```
PRINT TAXRTP.MYGROUP.ANYACCT
```

> Access to a file is governed by the security provisions in effect on your computer. While you may be able to find a file, you might—or might not—be able to see its contents, copy it, use it, or change it.

File names and logons

Have you noticed something a little odd?

The fully qualified name of the file called `MYREPORT` is this:

```
MYREPORT.MYGROUP.ANYACCT

file name.group name.account name
```

Nevertheless, in order to log on to the group in which this file is found, you would enter this:

```
HELLO JOHN.ANYACCT,MYGROUP[Return]

HELLO user name.account name,group name
```

The user name and the account name are the two most critical elements of the logon. The group, which is only part of an account, is added to the logon almost as a qualifier.

Types of files

Files may be written in *ASCII* form. Typically, these are text files of the sort that you might create while writing a letter or a report. If you use the `PRINT` command to examine them, they appear on your screen as you recorded them, in recognizable letters, numbers, and symbols.

Other files are written in *binary* form, the language that computers truly understand. Programs are usually recorded in binary form, and so are illustration files that are called *graphics*. Binary is the recording form for most databases, too.

Still other types of files are called *byte stream files*. These files are simply a sequence (stream) of bytes. The term *byte stream file* is frequently used when talking about files with the byte stream record type, even though they are not a new file type. *Byte stream files* do not have any record structure associated with them and have a record size of 1 byte. Also, they allow reading and writing of data in arbitrary chunks.

Execution or information

A last distinction among files is between *executable* files (programs) and *data* files (all others).

Executable files provide instruction for the computer to do something. Executable files are usually in binary form, although some, such as command files and job files are written in ASCII.

Data files hold information. Data files may be in ASCII or in binary form.

Mnemonics

The possible variations in file structure, type, and purpose are numerous. At the time it is recorded, each file is assigned a number or a *mnemonic* code. A mnemonic is a clue, usually a name or some descriptive word, designed to help you remember something. These serve to identify how the file was created and for what purpose. The code number for the EDIT/3000 program is `1029`. Its mnemonic code is `PROG`, because it is a program.

The File Connection

> For the computer, every source of information is a file.
>
> For the computer, every destination for information is a file.
>
> For the computer, groups, accounts and directories are all files.

As curious as it might sound, the computer looks at your keyboard as a file—an input file. And it looks at your video screen as a file, too—an output file.

Devices

Devices such as printers and your terminal (keyboard and video screen, together) belong in the story of files.

From the perspective of the computer, there is very little difference between a printer and a file in which you store information on a disk. Both are destinations for information.

One distinction is critical. You can send information to a disk file and retrieve information from it. But you can only send information to a printer. You cannot retrieve information from it.

Printers and terminals are so strikingly different in their physical operation that the computer does not attempt to "know" how to send information to these different destinations.

Instead, once it determines that information is destined for a printer, the computer passes the information to a system process, often called a *device driver*. It is the driver process that "knows" how to put information onto paper.

In the same fashion, once it determines that information is destined for your video screen, it passes that information to a different device driver (process), one that "knows" how to put information onto a screen.

In a similar fashion, you might take the blueprints for a house that you want to build and give them to a carpenter or a building contractor who knows how to

build houses. If all goes according to plan, you have nothing more to do. The carpenter/contractor "process" takes over, and, following the blueprints, builds a house. Processes and device drivers serve the computer in the same way.

The computer must have a special device driver for each kind of device that it uses—in fact, two printers that operate differently would need different device drivers. This touches on the subject of *compatibility*, the ability of two different computers to work with each other, or the ability of any computer to work with the devices that are attached to it. If two computers cannot adapt to each other's processes, they cannot communicate.

$STDIN and $STDLIST

Your terminal is, in fact, two files, `$STDIN` and `$STDLIST`. That is how the computer sees it.

These files are different from the letter or report that you might write and keep on the disk or print on a printer. These names will appear on your screen if you use the `SHOWME` command. What you will see looks very much like this:

```
:showme
USER: #S2367,JOHN.FINANCE,REPORTS          (NOT IN BREAK)
RELEASE...
CURRENT...
LOGON:...
CPU SECONDS: 11            CONNECT MINUTES: 56
$STDIN LDEV: 24            $STDLIST LDEV: 24
...
...
```

As file names go, `$STDIN` and `$STDLIST` look odd because the file names begin with `$`.

Only the computer system can create or use file names that begin with `$`. These "dollar sign files" are *system defined files* that the computer uses for sending and receiving information.

- `$STDIN` is the computer's *standard input file*.

  During a session, it is the keyboard that sits on your desk.

- `$STDLIST` is the corresponding *standard list file*, and it is the computer's standard output file.

The File Connection

During a session, it is the video screen on your desk.

When you enter information through your keyboard, it goes to the standard input file. `$STDIN` is a holding place for everything that you enter at the keyboard. As you type, `$STDIN` accumulates the characters that you type. When you press Return, the contents of `$STDIN` are sent to the operating system for analysis and action.

When the operating system sends information back to you during a session, it sends it to `$STDLIST`, your video screen.

The LDEV connection

You may have noticed that in the previous example of `SHOWME` that `$STDIN` and `$STDOUT` are both associated with something called LDEV: 24. LDEV 24 is the identity of a terminal.

It is likely that several people are using the computer at the same time. In order to distinguish among the constant flow of inputs and outputs, the computer has to know which terminal is sending information and which terminal should receive its corresponding output.

The computer solution is to assign each device a unique number. Your terminal device, for instance, might be identified as LDEV 24. LDEV stands for *logical device* (number). Device number 24 (a video screen and a keyboard) is "associated" with—"connected to" may be easier to envision—`$STDIN` and `$STDLIST`. And that is reflected in the display on the screen when you use `SHOWME`.

```
...
$STDIN LDEV: 24           $STDLIST LDEV: 24
...
```

This connection tells the computer the source and destination of information coming from and going to your terminal, or from any terminal. And because each terminal has a unique device number, the computer never becomes confused.

> A *logical* device is not a machine, nor is it a physical object. It is a connection to the computer that is made through an intermediating file or process. This "logical" connection is not the same as the cable used to carry information to and from a device. The cable is hardware, a physical object, a thing that you can touch.

The console

Whether your terminal is designated as 24, 25, or 37—or some other number—is determined when the computer is first set up. If terminals are added to the system, they will be numbered in sequence as they are added.

One terminal, however, is almost always LDEV 20: the console. The console is the terminal at which a system supervisor or system manager works to oversees the day-to-day operation of the computer. There is no special meaning to the number 20. It is simply a convention. If you wish to discover the LDEV number of the console, enter:

```
CONSOLE[Return]
```

You may, if you wish, think of the console as the master terminal for the computer. Certain commands and operations are permitted only at the console. Some of those commands and operations are mentioned in Chapter 7.

> A terminal is a physical object, hardware. In that sense, it is more accurate to regard the console as a terminal at which certain, critical operations are permitted. The person who manages your system may use the `CONSOLE` command to transfer those special abilities to another terminal. Using the `CONSOLE` command in this fashion is sometimes called "moving the console," although there is no physical motion involved.

Other LDEVs

If you use your computer's printer, you are (probably) using LDEV 6. A tape drive is usually identified as LDEV 7.

Certain LDEV numbers are assigned for special purposes. LDEV 1, for instance, is always the master volume of the system disk. That never changes. With such exceptions in mind, it is possible, for someone authorized to use a special program, the SYSGEN utility, to assign almost any LDEV number to any device. Such assignments are made when your system is first set up.

> Using SYSGEN requires special knowledge. Do not attempt to use it without proper training.

LDEV 6 is *probably* associated with your printer. LDEV 7 is *usually* associated with a tape drive—but there is nothing to prevent someone from connecting your printer to a different LDEV. It is certainly worth asking the person who manages your computer to tell you which LDEV numbers are assigned to which devices. Or you may use the `SHOWDEV` command to see a listing of device assignments.

The file equation connection

If your work involves using programs, you can usually rely on the program that you are using to direct information to a printer (or to another device) when you issue the proper instructions. These instructions usually take the form of commands that are "built into" the program itself.

The commands built into any program "talk" to the computer. In essence, a command might say to the computer: "I have something to print. Please send this to the correct place." The computer looks among its resources, finds a printer (if there is one), and directs the information to a printing process. Under these circumstances, the computer "chooses" the printer that will receive and print the information.

> The computer does not make whimsical choices. Behind every action that the computer takes is a program or process that makes choices based upon the information that it receives from you and upon information that it has about its own resources.

However, there may be times when you may want to influence the choices that the computer makes.

The PRINT command normally instructs the computer to display the contents of a file on a video screen. That is how the programming engineers designed it.

You might, however, want to use the PRINT command to send the contents of a file to a printer. You can do that.

> From the computer's point of view, "printing" something on paper, and "printing" something on the screen are virtually identical tasks, because it regards printers and video terminals as files.
>
> This makes things simpler for the computer, but sometimes makes them a little puzzling to people.

When you use the PRINT command the computer's "thinking" goes something like this:

```
"I AM GOING TO TAKE THE CONTENTS FROM ONE FILE"
"I AM GOING TO SEND THE CONTENTS TO ANOTHER FILE"
```

From which file? To which file?

The computer does not care. Instead, it takes its instructions regarding to "from" and "to" directly from the PRINT command.

```
PRINT TAXJAN
```

That seems understandable: print the contents of the file TAXJAN.

The computer understands that, too, and it displays the contents of the file TAXJAN on your video screen. Why?

PRINT TAXJAN specifies a file for the computer to find. It does not *explicitly* specify any destination to which it should send the contents of that file.

Still, the computer knows what to do. During a session, the PRINT command is programmed to send information from a disk file to the system-defined file called $STDLIST, which happens to be your video screen. Because no

destination was explicitly specified with the PRINT command, the computer uses the *default* destination that it was programmed to use when you do not explicitly specify a destination ($STDLIST). This touches on the subject of default parameters, which is covered in "Defaulted or Omitted Parameters" in Chapter 7.

If instead you wanted to use PRINT to send the contents of a file to a printer, you would need to instruct the computer to send it to a file *associated with* or *equated to* a printer.

In essence, you tell the computer: "I want you to behave as though the printer is the real destination for my file" This *redirection* of information involves using the FILE command this way:

```
FILE XYZ;DEV=LP(Return)
```

Using the FILE command in this way creates a *file equation*. LP is a *device class name* for a printer. This use of the FILE command tells the computer: "during this session, please regard the file XYZ as a printer."

> The device class name for a Mercedes Benz is "automobile" (or "car"). A device class name identifies things that are sufficiently alike in their appearance or operation that a single word adequately describes them. There may be differences between them, as there are differences between automobiles, but the differences do not alter the fundamental "sameness" of their design, use, or purpose.

Where is the file XYZ, and what is in it? In purely physical terms, it does not exist, nor is there anything in it. You might think of it as an alias, an invented name that refers to a real person but not by that person's real name.

To print a file called TAXJAN on the printer known by the device class name LP, you would next use the PRINT command this way:

```
PRINT TAXJAN,*XYZ(Return)
```

This "sentence" of computer instruction tells the computer to send the file TAXJAN to a file (an alias) called XYZ.

The asterisk (*) preceding the XYZ creates a *backreference*. The computer then looks back to find the file equation in which XYZ is defined (FILE XYZ;DEV=LP Return).

When the computer traces this reference to XYZ, it finds that XYZ is to be regarded as a printer. The file called TAXJAN goes to the printer, just as you wanted it to do.

> *Worth Remembering*
>
> LP is a common device class name for a printer on an MPE/iX system; however, the people who set up your computer could—*and might*—assign a different device class name for your printer(s).

It is possible—and very likely—that someone has already created a file equation to direct information to your printer and has made it a permanent feature of your system. To see a listing of the file equations that are current during your session, enter this command:

```
LISTEQ Return
```

Whatever device class name is assigned, many of the programs that you use, including HP Easytime/iX, will already "know" how to find the printer and how to send information there. Whatever device class name is assigned, it may be worthwhile to ask for that information from the person in charge of your system.

Input default for PRINT

During a session you might, in a moment of haste, enter this:

```
PRINT Return
```

Now the computer will obediently take information from the system-defined file $STDIN, which happens to be your keyboard. And it sends whatever you type to the system-defined file $STDLIST, which happens to be your video screen. The results can be puzzling.

Nothing happens until you type a line at the keyboard. Then, when you press Return, the computer sends that line to your video screen. What you will see is something like this:

```
MARY HAD A LITTLE LAMB, ITS FLEECE WAS(Return)
MARY HAD A LITTLE LAMB, ITS FLEECE WAS
-
```

The problem now is that the computer will go on doing this until you tell it to stop by using the :EOD (end of data) command this way:

```
MARY HAD A LITTLE LAMB, ITS FLEECE WAS(Return)
MARY HAD A LITTLE LAMB, ITS FLEECE WAS
:EOD(Return)
-
```

You must enter EOD on a blank line, and you must put the colon (:) before it.

> PRINT ,NEWFILE (notice the comma) takes whatever you type at the keyboard, line by line, and puts it into a file called NEWFILE, until you enter the :EOD command. NEWFILE is a temporary file, meaning that it will disappear when you log off. To turn it into a permanent file, enter this:
>
> ```
> SAVE NEWFILE(Return)
> ```

Redirecting STORE

Redirection (through a file equation) is involved in using the STORE command to copy files from the disk to your DDS tape:

```
FILE T;DEV=TAPE
STORE @.@.FINANCE;*T
```

Here, TAPE is a device class name for a tape drive. All of the files in all of the groups in the FINANCE account will be copied to tape.

A file equation of this sort may have been established on your system. T is a common reference in MPE/iX for a tape drive. Still, it *might* be something different on your system. That, too, is worth knowing.

Configuration—teaching the computer

The number of devices connected to your system is potentially large. With some exceptions, your computer leaves the factory having no knowledge of which devices you might attach or how many of them.

Telling the computer which devices are attached (and how to find them) is a process called *configuration*. The configuration—a description of the number, type, and location of its parts—might differ from one MPE/iX computer to another. You might attach devices that someone with another MPE/iX computer does not use at all. One computer system might have six terminals; another might have 23.

When your computer is first set up, SYSGEN is one of the very first commands that the manager of your system uses. The SYSGEN command calls up the SYSGEN utility, a highly specialized program that starts the configuration "dialog."

With your Series 9X8LX computer, you may, if you wish, have SYSGEN draw upon default settings to define the devices attached to the system.

At the end of the configuration dialog, that configuration information is stored in a *configuration file*, which is one of the *system files* that the computer maintains for its own use.

> Using SYSGEN requires special knowledge. Do not attempt to use it without proper training.

System files should be backed up to a special *system recovery tape* (*SRT*) when the system is first set up. And it is wise to back up these system files whenever they change. They *can* change.

Each time you add one or more new devices to your system, you must use SYSGEN to add this information to the existing configuration. This is the time to back up the system files and create a new SRT. In the event of a serious problem with the computer, you might have to reconfigure the entire system. Having a current set of backup system files makes the job much easier.

Spool files/print files

One more significant group of files deserves your attention. These files exist because no computer system can do all things simultaneously.

Spool files are different from the kinds of files that you will work with directly. In fact, most of the time you will have little contact with spool files, except under special circumstances.

Spool files exist to hold information that is on its way to somewhere else, such as printer. On the instructions of a program or a command that you might use, the computer creates spool files. They are the computer's equivalent of in-baskets and out-baskets. The computer fills them, waits for an instruction to use the contents, sends the contents on for processing or printing, and then (usually) erases them.

For that reason, the story of spool files belongs in Chapter 5, under "Spool Files/Print files" in Chapter 5.

4

Here I Am—What Can I Do?

The question is actually a little more involved. You need to ask: "How am I logged on? Where am I? What can I do?"

The answers to these questions depend upon these considerations:

- the user name of your logon
- the group to which you logged on
- the account to which this group belongs

Capabilities

Your ability to use the computer's many facilities may be extensive, or it may be limited. The extent of your ability to use the computer's facilities depends upon an MPE/iX security concept known as *capability*.

> Capability, in this sense, does not reflect any judgment made about the person using the computer. Capabilities are levels of authority (or permission) needed to use various functions that are available within the computer.

A computer allows you—or your organization—to gather together all of the information that is needed in one compact form and make that information readily available to anyone who needs it.

At the same time, *some* information that your organization uses is sensitive—personnel records, for example. Some information—and the means of acquiring and using that information—needs protection, above and beyond the security provided by passwords and file security.

Some commands and programs on your computer are vital to the efficient and orderly operation of the computer itself. Such commands and programs must not be misused. Their misuse can disrupt the operation of the computer itself, and can, in some cases, cause important information to be lost or destroyed.

Capabilities protect the computer itself—as well as programs, functions, and vital data—from misuse.

Who Can, Who Cannot . . .

Capabilities are *assigned*, and they determine the extent to which you, or any other user, can make full use of the computer's facilities. Capabilities are assigned to these elements:

- users (user names)
- accounts (account names)
- groups (group names)
- certain commands, programs, and processes within the computer

On most MPE/iX computers, you will find that there is a hierarchy of capabilities. Some users have more capabilities, or more powerful capabilities, than others.

This arrangement of capabilities can be tailored to meet the needs of your organization.

As a generalization, the range of capabilities, from low to high, is likely to be something like this:

- Someone who can log on to only one group in an account *generally* has the smallest set of capabilities.
- Someone who can log on to several—or all—of the groups in an account *generally* has a larger set of capabilities.
- Someone who can log on to several different accounts *generally* has an even larger set of capabilities.
- Someone who can log on at the console as `OPERATOR.SYS` or as `MANAGER.SYS` has the largest set of capabilities.

> It is possible to give every person using the computer *almost* full and equal capabilities on the system.
>
> The potential for confusion and disruption of the system is too great to make this practical or desirable. Equally troubling, such an arrangement would reduce the security of files and programs on the system.

In charge of the entire system

At the highest level, capabilities are assigned by the person who plays the role of *system manager* on your system. This is the person who knows how to log on as `MANAGER.SYS`. Strictly speaking, the system manager is not a person. Rather it is a user (user name) that has been granted system manager capability within the `SYS` account. The computer code for this capability is *SM*.

As you might infer, the user that has SM capability has a high "status" on the computer system and has access to the widest range of programs and functions. It is the system manager who creates accounts and assigns them passwords. The system manager can log on to any group in any account on the system and is able to discover the passwords assigned to every account, every group, and every user. In addition, the system manager has the authority to analyze and fine tune the performance of the system.

In charge, day to day

A less extensive set of capabilities is assigned to the user (user name) who plays the role of *system supervisor* or *system operator*, for which the computer code is *OP*. This user can log on as `OPERATOR.SYS`.

The system operator has day to day responsibility for the functioning of the system. That may include the creation of accounts. If yours is a small-size or medium-size organization, the roles of system manager and system operator may be combined in one person (user).

Other tasks that fall to `OPERATOR.SYS` may include these:

- monitoring the console to read and respond to messages from users

- controlling the efficient use of peripheral devices, such as printers, disk drives, and the like
- managing sessions—allowing or restricting activity on the computer
- managing jobs
- backing up files and restoring them
- starting up and shutting down the system (when that is necessary)
- trouble-shooting problems

In charge of an account

Every account on the system has an *account manager* (code *AM*). AM capabilities are different from OP capabilities, but they include such things as the creation of users and groups within the account, and the assignment of passwords and capabilities to users and groups within the account.

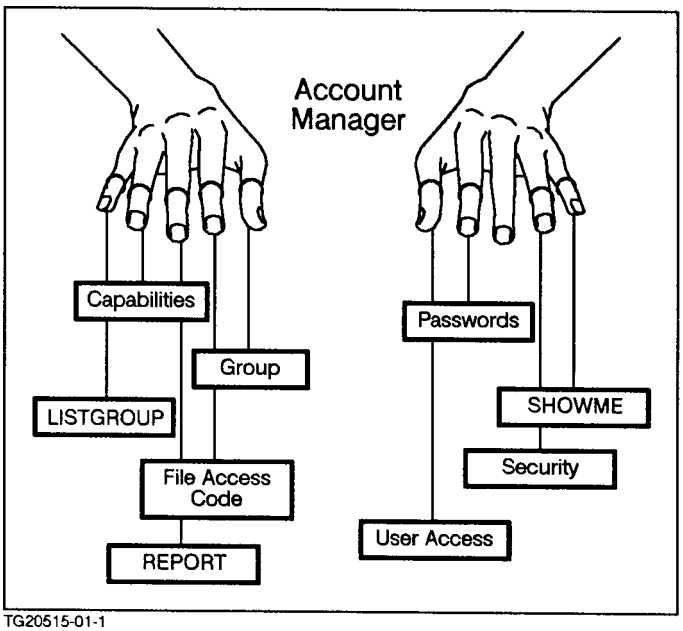


Figure 4-1. Your Account Manager

Who Can, Who Cannot . . .

If yours is a small-size or medium-size organization, the system operator or system manager may serve as the manager of one or many accounts.

At the group level

Users (user names) that are assigned no extensive capabilities acquire the abilities that are assigned to the group (within the account) to which they log on.

As a generalization, groups have the least extensive set of capabilities in the hierarchy of capabilities. However, the creator of the group (the account manager, system operator, or system manager (AM, OP, or SM capability) determines the capabilities that the group will have.

It would make little sense to give to a group capabilities that are greater than the capabilities of the account in which it resides. In fact, it is impossible to do so.

Users

For users, these are the default (standard) capabilities:

Code	Capability
IA	*Interactive Access* This capability allows you to log on and start a session on the computer.
BA	*Batch Access* This capability allows you to start (batch) jobs on the computer.
SF	*Save Files* This capability allows you to save your work in a file on the disk.
ND	*Nonshareable Devices* This capability allows you to send information (files) to printers, tape drives, and other nonshareable devices. These devices are described as "nonshareable" because they accept only one file at a time.

If you log on in . . .

Imagine an account called MYACCT. One of its users is JOHN. Imagine that this user happens to have only the standard (default) capabilities. Imagine, too, that this account has these three groups and that they have minimal capabilities:

- PUB

  This is the shared group in MYACCT. The logon is: HELLO JOHN.MYACCT,PUB.

- MYGROUP

  This your home group. The log on is: HELLO JOHN.MYACCT. (You can log on to your home group without specifying the group name.)

- OTHERGRP

  Some other group. The log on is: HELLO JOHN.MYACCT,OTHERGRP

Finally, suppose that there is a file called MYREPORT that you can put into any one of these three groups to show what happens.

The tables that follow illustrate what you can and cannot do with MYREPORT if you log on to any one of the three groups.

In each case, you could try to do these things:

- Read the file (the TEXT command found in EDIT/3000);
- Save the file (the KEEP command found in EDIT/3000)
- Erase the file (PURGE)

User JOHN logs on in PUB

If MYREPORT is in this group	Read file from PUB?	Save or Erase file from PUB?
PUB	Succeeds	Succeeds
MYGROUP	Succeeds	Succeeds
OTHERGRP	Fails	Fails

User JOHN logs on in MYGROUP

If MYREPORT is in this group	Read file from MYGROUP?	Save or Erase file from MYGROUP?
PUB	Succeeds	Fails
MYGROUP	Succeeds	Succeeds
OTHERGRP	Fails	Fails

User John logs on in OTHERGRP

If MYREPORT is in this group	Read file from OTHERGRP?	Save or Erase file from OTHERGRP?
PUB	Succeeds	Fails
MYGROUP	Succeeds	Succeeds
OTHERGRP	Succeeds	Succeeds
another group	Fails	Succeeds

Other capabilities

There are other capabilities in addition to the ones mentioned thus far. Many of these other capabilities serve to restrict the use of very specialized user names, programs, and processes. In particular, two special capabilities prevent the unauthorized use of sensitive or extremely powerful programs: *PH* (process handing) and *PM* (privileged mode).

Whoever manages your computer system, or your account, should be sure to give you the capabilities you need to do your day to day work. In general, the specialized and powerful capabilities are reserved for those who must manage or run your computer system, and in some cases for those who manage accounts. Unless you have such responsbilities, you are not likely to need specialized capabilities.

Access control definitions (ACDs)

ACDs are ordered lists of pairs. These pairs consist of access permissions and user specifications that control the ability to access and change MPE files, hierarchical directories, and the files within them. ACDs are applied using the `ALTSEC` command and take precedence over other security features, such as lockwords and file access control, such as read/write access.

For more information on ACDs, refer to Chapters 3 and 9 of the manual, *New Features of MPE/iX: Using the Hierarchical File System*, (32650-90351). For more information on the `ALTSEC` command, refer to the *Commands Reference* (B3813-90011).

User and Group IDs

Each MPE/iX user has an associated **user ID (UID)**. The UID is a string (in the form *user.account*) and a corresponding integer value. Additionally, one or more users can be organized into groups (distinct from MPE groups) to simplify file sharing. Each group has an associated **group ID (GID)**.

UIDs and GIDs are used in conjunction with other security mechanisms to control access to objects. **Objects** are entities that contain or receive information, such as files, directories, and devices. When files or directories are created, they are assigned their parent directory's GID and the UID of the process creating them.

UIDs and GIDs are stored in two databases: HPUID.PUB.SYS holds UIDs and related user information in a **user ID database**, and HPGID.PUB.SYS holds GIDs and related information in a **group ID database**.

Can I . . . ?

You probably can!

Programs

Even with the minimum set of capabilities, you will be able to run any program that is not protected in some way. Any program found in `PUB.SYS` (the `PUB` group of the `SYS` account) is available for your use, as well as any other files that reside there.

Unless they are protected in some way, you will be able to run any program in the shared group of your logon account (it might be the `PUB` group). The programs that are in your home group are available to you, too.

Protection might consist of putting a program into a group or an account to which you are unable to log on (because you do not know the correct passwords). It might consist of attaching PH or PM capability to the program.

You will have access to most of the built-in MPE/iX commands. In addition, you have at your disposal *command files* and *jobs*. Each of these facilities serves a purpose, and at one time or another, each will prove valuable. Each is discussed in Chapter 8.

This guide does not attempt to instruct you in the use of any of these facilities, but, rather, defines and explains them as tools that you may use. For details on how to use these facilities, turn to the *Task Reference - HP 3000 Series 9X8LX* (B3820-90005).

Commands

With the exception of those commands reserved for the console, you will be able to use almost all of the commands built into MPE/iX. There are commands, however, that are reserved for those users who have special capabilities. The `NEWGROUP` command, which creates new groups within an account, is available only to an account manager (AM capability).

In addition, some commands will perform at full power only for users who have special capabilities. `LISTACCT` shows the names of all of the accounts on the system. You can use this command to see the names of all of the accounts that exist. `LISTACCT` *acctname*;`PASS` shows the name of an account and its password. This use of the `LISTACCT` command works only for the system operator (OP) or the system manager (SM).

Starting Programs

> The documentation that accompanies each program explains what it does and how to use it. In particular, the use of HP Easytime/iX is presented in detail in Volume I of *Using Your System*.

You have three ways of starting—sometimes called *running*—any program on your MPE/iX system.

- starting a program with the `RUN` command
- starting a program *without* the `RUN` command
- starting a program with the `XEQ` command

It might seem odd to have three ways of starting (running) a program. On earlier MPE systems, the only way to start a program was to use the `RUN` command. As the system evolved, programming engineers added other methods, to provide more flexibility and to manage the growing complexity of the system and its facilities.

Which method you use is *usually* a matter of choice. However, under some circumstances, one method will work, and another will not.

Starting with the RUN command

`RUN` is one of the commands built into the MPE/iX operating system. Its chief purpose—but not the only one—is to start a program.

> You may use the `RUN` command to start any program that exists on your system—provided that the program is not restricted from general use by some method of protection.

Consider a program called AUDIT that resides in the REPORTS group of the FINANCE account. Its fully qualified name is AUDIT.REPORTS.FINANCE.

> Notice that programs are files, too. They have fully qualified names: AUDIT.REPORTS.FINANCE.

Wherever you log on, you can run AUDIT this way:

```
RUN AUDIT.REPORTS.FINANCE
```
Return

This way of starting a program works for any program, *if* you know the fully qualified name of the program file, and *if* the program is not restricted from general use by some method of protection.

Notice, though, that

```
RUN AUDIT
```
Return

will not work, unless you are logged on in the group where the AUDIT program is found (in the REPORTS group of the FINANCE account in this example).

Starting without the RUN command

However, *if* you are logged on as JOHN.FINANCE,REPORTS, you can run AUDIT this way:

```
AUDIT
```
Return

This "first name only" way of running a program is called an *implied* RUN.

> This second method of running a program may not work in each and every case.
>
> Before the computer can start a program, it must first *find* the program on the disk. Using `RUN` with the fully qualified name of the program tells the computer specifically where to find that program.
>
> Using an implied RUN forces the computer to search for the program. To do that, it follows a *search path*

Path restriction

If you use an implied `RUN` (`AUDIT`Return), the computer follows a set of instructions called a *path*. These instructions tell the computer where to search for programs (or any executable file). If the program resides in an account (or in a group) *not* mentioned in the path instructions, the computer will *not* find that program.

By default the path instruction tells the computer to search in these places, in this order:

1. the group in which you are currently logged on
2. the `PUB` group of the account in which you are currently logged on
3. the `PUB` group of the `SYS` account

You can discover what path is set for you by entering this:

```
SHOWVAR HPPATHReturn
```

`HPPATH` is a system variable (an area of computer memory) that contains the path instruction. The `SHOWVAR` command displays the setting of this—or any other—system variable. `SHOWVAR @`Return will display the settings of all of the system variables for your session.

The `SETVAR` command allows you to change the path along which the computer searches for executable files. You may use the `SETVAR` command to change the value of a system-defined variable called `HPPATH`.

Starting Programs

If the program that you want to run is not found along the path instructions, you will need to use `RUN` and specify the fully qualified name of the program.

Name restriction

Still another limitation on using an implied RUN name might be the very name of the program you want to run.

The computer follows a set of priorities—a *search priority*—in its hunt for executable files. The priority of the search, from highest to lowest, is this:

- *highest priority:* user-defined commands (UDCs)
- *next priority:* built-in commands (see Chapter 7)
- *lowest priority:* programs and command files (see Chapter 8)

User-defined commands are somewhat like command files, but they are an advanced topic in using the MPE/iX operating system. You will find information about user-defined commands in *Using the 900 Series HP 3000:Advanced Skills* (32650-60039). You may order this book through your Hewlett-Packard representative. Other books that might be of interest are mentioned in the bibliography found in the book *Task Reference - HP 3000 Series 9X8LX* (B3820-90009).

A program called `MYPROG` will come into conflict if there is a user- defined command called `MYPROG` or if there is a built-in command called `MYPROG` (there is no such built-in command, by the way).

1. If the computer finds a user-defined command called `MYPROG`, the computer will execute that first and stop its search.
2. If, instead, the computer finds a built-in command called `MYPROG`, the computer will execute that first and stop its search.

Only after it has exhausted these possibilities will the computer continue its search for the program called `MYPROG`.

One of the MPE/iX built-in commands is called `SHOWME`. If there is also a program called `SHOWME` on your system, you will encounter difficulty in trying to run the program `SHOWME`. If you enter `SHOWME`Return, the operating system will execute the *command* `SHOWME` and fail to execute the *program* called `SHOWME`.

There are two solutions to this problem.

- One solution is to use the `RUN` command with the fully qualified name of the program (`RUN SHOWME.group.account`).
- The other solution is to use the `XEQ` command.

Starting with the XEQ command

If you wish to execute a program called `SHOWME` and not the command called `SHOWME`, you may use another command, `XEQ`, to execute the *program* `SHOWME`:

```
XEQ SHOWME(Return)
```

`XEQ` is a shorthand way of writing the English word "execute," and it has other applications that are discussed in Chapter 6. The `XEQ` command bypasses the user-defined commands and the built-in commands before searching for an executable file.

5

Where Does the Information Go?

A complete answer to "where does the information go?" would take many pages of explanation.

In part, the answer depends on what you are doing and what you want to accomplish. And in part, it depends on the methods by which the computer sends information from one place to another.

If you use HP Easytime/iX, you can point the way and have your information go exactly where you want it to go. HP Easytime/iX allows you to navigate through the computer and arrive where you want to go, without looking at a road map and without stopping too often to ask for directions.

This chapter is devoted to the question of where your information goes inside the computer. In a very real sense, the activities of the operating system, of all of its commands, and of all of the programs you use *are* the answer. But that answer is written in several million instructions that direct the computer in its interactions with you. That sort of road map for information is complex in the extreme.

Instead, this guide looks at some of the highlights along the way.

To and From Your Terminal

Even before you log on, you can begin a "dialog" with the computer. That might seem an odd way to describe what is going on. The computer is not a person, and its vocabulary is severely limited compared to yours. Yet, an exchange of information is taking place.

Before you log on, the computer displays a *log on prompt*:

```
MPE/iX:
```

That is the computer's message: "I am waiting for an instruction." In this case, it is waiting for you to log on using the `HELLO` command.

After you log on, the computer displays a *system prompt*:

```
:
```

This message says "I am the operating system. What do you want to do?" Now you have at your disposal all of the built-in commands and whatever programs may be on your system that fall within your *capabilities* as defined by your user name, account name, and group name.

An exchange of information goes on every time that you press a key on your keyboard. Every key sends (as output) a number code (as input) to the computer.

Pressing the A key is input to the computer. If you press A and do *nothing else*, the key code for "A" is sent to the computer. The computer examines the code, determines that it is a letter, and acknowledges your input by sending back (*echoing*) a letter "A" to your video terminal. This happens so quickly that it appears instantaneous.

Your part of this dialog consists of typing (input) "A".

The computer's part of this dialog is: "I received something from you. Here is what I understood you to say."

Until you press Return, nothing beyond this simple input and output happens.

> [Return] says "Please interpret and carry out my instruction if you can."

If you now press [Return], something happens: the computer attempts to find the significance of whatever you have entered.

If you are at the system prompt (:), the computer might—or might not—recognize what you typed. There is no built-in command called `A`. Unless someone has instructed the computer to accept "A" as being somehow significant, the computer will display an error message on your video terminal.

In general, an error message—there are many kinds—says: "I received something from you. I cannot understand it, or I cannot carry out your instruction. Here is what puzzled me."

> *Worth Knowing*
>
> If your keyboard does *not* have a key labeled [Return], the same function is performed by the [Enter] key. If this is the case, be sure to *think* "[Enter]" when you see [Return] in this guide.
>
> If your keyboard has *both* [Enter] and [Return], the [Return] key is probably the key that tells the computer to accept what you type at your terminal. Computer keyboards do differ. If you are in doubt, look in the documentation that describes the operation of your terminal and find which key sends information to your computer.

Pressing [A] alone produces a simple response by the computer. Following [A] with [Return], however, triggers a series of processes by the computer. The computer examines everything that preceded [Return] and attempts to determine its significance.

To do that, the computer examines several areas of its own memory to find the meaning of "A". If it cannot find any meaning, it displays an error message on your video screen.

You could, if you like, create a command file called `A` that *would* have significance for the computer. Once you did that, `A`Return would be a command to the computer to execute the instructions in the command file that you created and called `A`. Command files are discussed in Chapter 8.

`A` is not very informative to human beings. Unless you are certain of remembering what you put into `A`, it would be better to call it something else, perhaps `MYFIRST`—something that would jog your memory about what the command file does.

What computers "know"

> Computers "know" only what they have been told. They are able to do only what they have been instructed to do.

When this guide speaks of the computer "knowing" or "understanding," it means only that someone has programmed the computer to react in a useful and predictable way to a given set of circumstances. Whoever has done that programming has anticipated those circumstances and has created rules by which the computer is able to determine its course of action. When the computer encounters a set of circumstances for which it has no rules to follow, it is likely to "hang" (to become temporarily paralyzed) or confused, and it may display a message indicating its confusion about your intentions.

There are experimental computers that do "learn." Among them are neural network computers that, in their design, attempt to mimic the activity of the human brain. Using artificial intelligence (AI) programs, they are able to add to or modify the rules (programs) by which they are governed and, consequently, are able to alter their own behavior over time. They are *not* electronic brains—although their design is modeled on the human brain—nor are they intelligent in any human sense of the word. Where such developments might lead is open to speculation.

Most computer programs can answer only "Yes" or "No" to questions that involve comparison: Does X equal Y? (Yes or No)—Is X bigger than Y? (Yes or No)—Is X smaller than Y? (Yes or No). They thrive on precision.

Artificial intelligence programs attempt to instruct the computer in the sort of vague comparisons that human beings make all of the time: *almost*, *not quite*, *more or less*, *nearly*, *a little*, *many*, *few*, *big small*, *probably*, *maybe*. Human beings use these sorts of comparisons when we make decisions that are based not on precise information but on estimations, guesses, or hunches.

Understanding your command

The scanner and parser are programs (to be accurate, they are parts of a program). Like any program, they are recorded instructions that the computer has been told to follow in well-defined circumstances.

Together the scanner and parser and the *command interpreter* (CI) examine what you have entered. For the computer, `HELLO BUDGET,JOHN.FINANCE,REPORTS`Return constitutes a complete sentence of instruction. Return terminates the sentence. Spaces, commas, periods, slash marks (/), and other symbols are the punctuation of the sentence.

> *Worth Remembering*
>
> You can examine a sentence on a page and make sense of it, even if all the punctuation is removed. It may take a moment to sort it out, but you can do it.
>
> However, exact punctuation is indispensable for the computer. The computer does not have your ability to make an educated guess to determine the meaning of an instruction.

The punctuation tells the parser where each word in the sentence begins and ends. The computer then examines each word to determine its meaning and significance.

The computer will look in its own memory to determine whether HELLO is a command or some other instruction that it can execute. In contrast, HELLOW is a misspelling. More significantly, the computer would not find in its memory any command called HELLOW and would conclude that you had made an error of some kind.

- The computer will accept BUDGET as session name.
- It will look in its own memory to determine whether a user called JOHN is on its list of users to accept. At the same time, it will make note of the capabilities attached to JOHN.
- It will look to determine whether an account called FINANCE exists and whether the user JOHN is attached to the FINANCE account. At the same time, it will make note of the capabilities attached to FINANCE.
- It will look to determine whether a group called REPORTS exists. At the same time, it will make note of the capabilities attached to REPORTS.

If the computer determines that all of the names (and any passwords) are correct, it will accept your logon command.

Acting on your command

Now what?

The command interpreter does not by itself cause your instructions to be executed. Its purpose is to make sense of what you enter at the keyboard and then to send that information to the operating system, of which the CI is part. The operating system is the central and indispensable program in any computer. All other programs that you might use are dependent upon—and, in a sense, are subordinate to—the operating system.

At the heart of a computer is the processor. In MPE/iX computers, it is called the *system processing unit* (SPU). In some computers, it may be called the *central processing unit*. It is hardware, a computer chip, or a collection of computer chips. The processor counts, performs all arithmetic functions, and changes information from one form into another form.

Destinations for Information

Where information goes depends in part on where you want it to go and your reasons for sending it there. In part, where information goes is determined by the command or program that you are using and by the computer itself.

Computer memory

Computers have not one but several forms of memory. Which kind of memory is devoted to your work is determined by the stage at which you find yourself.

Random access memory (RAM)

If you are using a text editor, such as EDIT/3000, all—or certainly some—of your text will reside in the computer's *active memory* while you are working on the text. Active memory is called *random access memory*. At the same time, a certain portion of the computer's active memory is occupied by the text editor program itself.

When you start an editor program (by executing `EDITOR`Return, for example), the computer *loads* the program into random access memory and begins following the instructions that it finds in the EDIT/3000 program. If you then specify an existing text to edit (`TEXT REPORT1`Return—meaning "call up the file named REPORT1"), the program calls upon the computer to load the file called REPORT1 into random access memory as well. How much of your text is actually loaded into random access memory depends upon how much memory is available at the time. If you begin by creating new text, that text is placed into random access memory as you type.

Still other parts of random access memory are occupied by the computer's operating system and by the programs and information that other users are working on at the same time. In the background, the computer itself may load, run, and unload other programs in order to carry out its own tasks or the tasks that you and other users assign to it.

The amount of memory taken up by "background" programs and processes is called *overhead*. In a very real sense, overhead is an "expense" incurred by any program that you might use.

> Because random access memory in any computer is finite, specialized programs exist to "fine tune" the management of this vital commodity. In fact, one or more of these memory management programs may be running in the background while you are working on something of your own.

The active portion of the computer's memory exists only in electronic form and exists only while the computer is active (receiving electrical current and operating normally).

RAM holds those programs that you or the computer are running, as well as information (*data*) that the programs are working on—the text of your letter, for example.

Random access means that, with certain exceptions, material in RAM is available for your use at will. If you decide to insert a new line or a new paragraph in the text that you are writing, the editing program permits you to do that. By adding a new line or a new paragraph, you are inserting data into the computer's RAM memory. You may examine any portion of your text without having to start at the beginning and work your way down to the portion that you wish to see.

Disconnect or seriously interrupt the current, and everything in random access memory disappears. This is one reason that you will *almost never* turn off your MPE/iX computer. Equally important, turning the computer back on is not simply a matter of turning the switch from OFF to ON. All of the background programs and processes that the computer relies upon for its operation must be reloaded and synchronized. That takes time.

> You may turn off your terminal at the end of the day, because doing so does not affect the computer's operation.

When you log on, when you run programs, the computer devotes a portion of its random access memory to your session and to the commands and programs that you are using. It devotes some of its random access memory to every

user who is logged on to the computer. It reserves some memory for its own purposes, too. When you log off, the computer reclaims the portion of memory that was devoted to your session and your work and, if needed, reassigns that portion of memory to another session or program.

Random access memory is large, but it has limits. As more and more users log on, as they run more programs, or more complex programs, the computer can devote less of its time and memory to each user. At some point, the computer becomes so busy doing "a little of this and little of that" that users begin to notice that their work is slowing down. At this point, the system operator may have to step in and limit the number of jobs and sessions that are actively running, simply to allow the computer to reclaim some memory and to spend more time with the remaining sessions and jobs.

Your MPE/iX computer is well-designed to deal as efficiently as possible with a large volume of work. Still, you should be aware that on *exceptionally* busy days, your work might slow down, or your operator might impose limits. The operator might suspend some jobs—and resume them later. Or you might try to log on and find this message on your terminal:

```
CAN'T INITIATE NEW SESSIONS NOW
```

Read only memory (ROM)

Read only memory (*ROM*) is another form of computer memory. Like RAM, it is electronic. Unlike RAM, it is stored permanently inside the computer.

ROM is inaccessible. You can neither add to nor take away from the data that resides in ROM. It holds information that is vital to the computer's second-by-second operation.

Disk—saving information

Information that is vital deserves to be saved. When you have finished working on your information, you must transfer that information from RAM to a computer disk memory.

Almost every program has a command or a technique that permits you to save your information to disk. The EDIT/3000 program does this through its `KEEP` command: `KEEP REPORT1`Return instructs the program (and the computer) to move the information from RAM into a disk file called `REPORT1`.

If you leave a program without this vital, saving step, there is a chance that you will lose whatever you have added to RAM—the rest of that letter that you decided to finish writing today. Sophisticated programs will alert you and ask whether you really intend to leave the program without saving your latest efforts.

> *Worth Remembering*
>
> In the event of a loss of power or a catastrophic failure, information in RAM can be lost. Sophisticated computers, such as the HP 3000 Series 9X8LX, will attempt to save information from RAM to disk in the event of a catastrophe. EDIT/3000 saves information in temporary files that have names such as `K910805`. The numbers form a code indicating by year, week of the year, and day of the week, when the file was saved (1991, week 08, day 05).
>
> You should pause in your work from time to time and issue a save command through the program that you are using to put your information into a disk file.

Tape—protecting information

In comparison with RAM, disk memory seems almost permanent. Once recorded onto a disk, information is *generally* secure from loss.

Secure as it seems, disk memory is not truly permanent, nor should it be regarded as permanent—certainly not for information that you cannot easily replace or recreate.

> At some time, almost every person succeeds in accidentally erasing from the disk at least one file. Almost without fail, it is the one file that you need *right now*.
>
> If a file is important, store it safely on tape.

Almost every computer system provides some means for saving vital information in a safe, protected environment. Transferring, or copying, information from a disk to a magnetic or digital tape is a common and economical means. The process may be called *backing up* or *archiving*.

The term associated with MPE/iX computers is *storing*, from the command to perform this operation, `STORE`. Its complementary process, returning information from a tape to a disk, is called *restoring* (from the `RESTORE` command).

Still another reason for storing information to tape is economy. Disk memory, though large, is finite. Unless you can afford to purchase new disks and disk drives as you need them, you will eventually fill your existing disk(s) with information.

The more economical solution is to *store* files onto tape when they are no longer of immediate or near-term value. Then *erase* from the disk those files that are safely stored. The disk space taken by those files becomes free again. If that stored information is needed again, you may restore it to your system—provided you have kept enough free space on your disk(s) to accommodate those restored files.

Backing up files: full or partial

A *full backup* copies all user files on the system regardless of when they were created or last changed.

A *partial backup* copies only those files that have been created or changed since the last full backup.

> *Worth Doing*
>
> Establish and adhere to a schedule for backing up the files on your computer.
>
> Your backup schedule should include both types of backup. A typical backup schedule allows for one full backup one day a week, and a partial backup on each remaining work day.

When the person managing your system warns you that a backup is impending, heed any instructions to save the work that you are doing and, if requested, log off. During a backup, you will not be permitted to continue working or to log on and start new work.

The process of backing up depends upon having files that do not change during the backup. When the backup concludes, you will be allowed to log on again and resume your work.

Printers

For most of us, there is a value in having information on paper, if only to send that piece of paper to someone else. Many of us are simply more comfortable seeing something on paper, as well as on the computer screen. For that reason, a computer without a printer would lose much of its value to us.

Collectively, printers are considered one of many classes of *devices* that may be connected to a computer. Loosely defined, devices are those pieces of equipment that are connected to a computer that are not essential to the internal operation of the computer itself. Any device that is not inherently part of the computer itself is a *peripheral device*.

> Disk drives, tape drives, and terminals are peripherals, just as printers are.

You may attach one or many printers to your computer. Which kind of printer you choose depends upon the nature of your work and the degree to which you need (or want) to be able to create documents that approach the quality of printing obtained from a professional printer.

For many purposes, only rough or intermediate quality print is sufficient. For such purposes, *line printers* are likely to be adequate. Line printers take their name from their technique of printing one line at a time. *Page printers* print an entire page in one operation. For most purposes, *laser printers*, which are also page printers, provide the highest quality print definition.

Spool Files/Print files

Spool files exist to hold information that is on its way somewhere else—to your printer, for example. On the instructions of a program or a command that you might use, the computer creates spool files. They are the computer's equivalent of in-boxes and out-boxes.

In-box spool files are called *input spool files*. Out-box spool files are called *output spool files*. HP Easytime/iX refers to certain kinds of output spool files as *print files*. Print files are one kind of output spool file (files destined for printing).

The reason for creating spool files is that your computer cannot do everything all at once, fast as it is.

Your printer, for example, can print only one file at a time. More precisely, it can print only one character, or one line, or perhaps one page, at a time. While one line of your report is printing, all of the remaining lines in the report must wait. Every other file destined for printing must wait, too.

> *Worth Knowing*
>
> Spool files (input and output) are involved in many of the processes that the computer uses, not just in printing.

To avoid wasting time waiting for the printer, the computer creates spool files. The computer fills up an in-box or an out-box, then waits for an instruction to use the contents. When the opportunity to act arises, the computer processes its spool file(s) and moves on to the next task. Once it has finished using a spool file, the computer (usually) erases the spool file.

The inability of printers (and other devices such as tape drives) to handle more than one thing at a time makes them *nonshareable devices*. You and every other user on your system do share the *services* of the printer. But files destined for the printer (or some other processing) must often wait their turn.

The computer keeps track of all of the files destined for printing (or some other process). It records the day and time of day when they were sent for printing. It notes their size and their priority in the waiting line, called a *queue*. It

prints first those files of highest priority and prints them in the order that they were sent to the printer. It next prints files of lower priorities, again, printing them in their order of receipt.

Output spool files

The print file of a letter you write—an output spool file destined for the printer—may contain more than the original contents of the letter that you wrote and sent to the printer. This out-box spool file may contain formatting and printing instructions that turn the file into the finished, presentable letter that you had in mind. The instructions are added by the process that actually sends the file to the printer (frequently the program that you used to create your letter).

These out-box files are stored on the disk. They have unusual names, such as #O3488675, which signify, first that they are output (out-box) files—that is the "O" part of the name. The digits following "O" form a code that signify their order of creation.

> You may use the `LISTSPF` command to find the names of any output (and input) spool files that are connected with your session.
>
> Consult the `LISTSPF` command in the book, *Commands Reference* (B3813-90011) to learn how to identify the output spool file (print file) that is associated with your user name and account.

When its turn arrives, each output spool file destined for printing (a print file) goes to the printer. When the printing is finished, the computer usually erases the temporary output spool file (print file).

> An output spool file (print file) can be made permanent. Why would you do that? You might want to examine the file to be sure that it is, indeed, what you intended for the printer. You may wish to change its priority in the waiting line (the *queue*).
>
> The `SPOOLF` command allows you to make spool files permanent or to change their priority in the queue.

Once you have identified the output spool file created for your text file, you can do this, for example:

```
SPOOLF  #03488675;PRI=9
```

This will raise the output priority of the spool file `#03488675` to 9.

A print file is one form of output spool file—a file destined for the printer. There are other output spool files destined for other nonshareable devices (or awaiting processing of some kind), but these are not called print files. They are simply output spool files.

> It is tempting to give your output spool file the highest possible priority and race to the head of the line.
>
> Some work does truly deserve higher priority—but only some.

Input spool files

There are also *input spool files*. These, too, are waiting their turn, not for printing, but usually for processing of one sort or another. They are designated with names such as #I3499852—the "I" is for input.

As a generalization, input spool files contain instructions *for* the processing of information rather than the result *of* processing. Output spool files usually contain the result of some processing.

Unless you become extremely sophisticated in your use of the computer, input spool files will remain out of your reach, and you will have little reason to be

aware of them. You cannot examine or modify them. You can, however, find their names with the `LISTSPF` command.

Because the computer can process input spool files much more quickly than a printer can print output spool files, it is not uncommon for the input spool files created as part of your work to be processed and erased before you have a chance to find them.

6

Behind the Scenes

This chapter takes you a little further behind the scenes to explain more about how the computer handles information and why it operates the way it does.

There is no requirement to read this chapter, and if have no great interest in the how and why of computers, you may wish to skip ahead to the next chapter.

Why Numbers?

Computers do not truly understand words or letters the way we do. They understand only numbers. Everything that a computer can work with is represented inside the computer as a number.

To work with letters, computers assign numbers (codes) to each letter—in fact, to each key on the keyboard. When you press (A) on your keyboard, the keyboard electronically sends a number code to the computer. By examining the code number that it receives, the computer determines that it must display the character "A" on your screen.

> The *function keys* on your keyboard, labeled F1, F2, F3, ... , are a special case.
>
> Programs such as HP Easytime/iX store instructions in these keys. When they appear at the bottom of your screen, the function key labels tell you what function, if any, each key performs.
>
> Because different programs can store different kinds of software instructions in these function keys, they are sometimes called *soft keys*.

When you type words, the computer looks at each number key code that it receives, determines what to display, and obligingly strings together the characters that you typed.

> In general, what we regard as a word—a collection of letters strung together in a particular order that has meaning to us—has no intrinsic meaning to the computer. However, certain combinations of letters and numbers do have meaning for the computer—but only because a programming engineer has instructed the computer to behave that way.

"CAT" means something in the English language: a small, furry, four-legged feline that behaves in ways that we have come to recognize. To the computer, however, it is simply the characters C, A, and T bound together in what is known as a *string*.

When you enter `HELLO` at the logon prompt, that entry *does* have meaning for the computer, but it has meaning only because a programming engineer has instructed the computer to recognize this string of characters. Because it recognizes `HELLO` as a command, the computer takes action some action—it starts a session.

Computers are electronic machines, and the fundamental language of electronics is simple. The current is either On or it is Off.

> Imagine that you had to talk to another person and your only way of communicating was to turn a light on, or turn a light off. What would you do?
>
> Computers face the same problem.

You might decide to say very little, of course, or you might decide to create a code based solely on these two primitive messages:

- The light is on.
- The light is off.

Turning the light on might stand for "Yes." Turning it off might stand for "No." That leaves very little to say.

The inventor Samuel Morse faced the same sort of problem in trying to communicate with a telegraph. His solution was the Morse Code, a sequence of long and short pulses on the telegraph key. (There is an international telegraph code, too.) Each letter of the alphabet is assigned a sequence of long and short pulses. Computers do something very similar.

In computers, the solution has been to designate the number 1 to represent "On," and 0 (zero) to represent "Off."

Why Numbers?

The entire "alphabet" of electronic computers consists of two digits, 1 and 0. This limited "alphabet" has advantages for the computer and some obvious disadvantages for human beings.

The advantage for computers is that counting from 0 to 1 at the speed of electricity goes *very* fast. The disadvantage for us is that with an "alphabet" of only two digits, there is not much that you can say, and counting looks hopeless.

Arithmetic for humans became far easier with the invention of the decimal system of notation (the digits 1 to 9, plus a dot to represent zero), which we owe to the ancient Sanskrit mathematicians. Arabic mathematicians invented the zero and used the dot to separate the whole number from any fractional part: 11.1 is one *ten* and one *one* plus *one-tenth* of one.

Reading from *right* to *left*, 2013 is three *ones*, plus one *ten*, plus no *hundreds*, plus two *thousands*. There are only ten digits to work with (0 to 9) , but the *position* of each digit tells you the *magnitude* (size) of the number that it represents. As you move from right to left, each place represents a magnitude (size) 10 times its neighbor to the right.

Binary notation

Binary notation has only two digits, 0 and 1. That makes it ideal for computers. Using binary notation, computers can count very nicely—although it still looks a little odd to most of us. Binary comes from "bi," meaning "two."

With binary notation, moving from right to left signifies a magnitude (size) 2 times its neighbor to the right.

`Binary`	Meaning	Decimal Equivalent
`1`	one one	1
`10`	no ones, plus one two	2
`1011`	one one, plus one two, plus no fours, plus one eight	11
`101010`	The *leftmost* digit represents one 32.	42

Addition appears strange, but that is that you must work with twos instead of tens:

```
Binary    Decimal      Binary    Decimal

   101    5            101010    42
+   11    3         +    1011    11
------   --           ------    --
  1000    8            110101    53
```

Each digit, 1 or 0, (On or Off) is called a *bit*. The largest number that a computer can count depends on how many ones and zeros that it can examine at one time.

Older personal computers, for instance, might examine only eight bits at a time: `11111111` is the largest counting number that they can manage. A grouping of eight bits is called a *byte*. More powerful computers can examine 16 bits (two bytes) at a time. Others can examine 32 bits (4 bytes).

What about numbers such as 387.435? Computers work with fractional numbers, too. However, the method of representing such numbers inside the computer is genuinely complex.

The search for greater computer power is largely a search for ways of allowing computers to handle larger and larger numbers of bits (or bytes) in one "gulp"—hence, the rush to develop computer "chips" that can accommodate more and more bytes. (To be sure, other factors are involved, such as the speed at which each "gulp" of information moves.)

ASCII code

If you feel that binary code is awkward for human beings, you are not alone.

The solution that makes computers workable for us is the *ASCII* code (American Standard Code for Information Interchange). It is not the only code for computers, but it is in wide use. Your MPE/iX computer uses the ASCII code.

Like Morse code, ASCII code assigns to each letter—not a sequence of longs and shorts—but a sequence of bits (zeros and ones). Every key on your keyboard (every letter, every number, every symbol) is assigned an ASCII code. So is Return and every other key. Shift modifies the code sent by many keys.

The Return key, for example, does double duty. Its code tells the computer to interpret whatever you have typed. It also tells the computer to end one line and move to a new line and wait for something new, much the way Return does on an electric typewriter.

The binary code for A is `1000001`, and it is numerically the same as 65 in decimal notation. The first 64 codes represent digits (0 through 9), the special symbols on the keyboard (! and # and & and so on), and special codes that are meaningful only to the computer.

All of the letters of the alphabet in English and most in European languages can be represented in a single eight-bit byte. There are enough different possible combinations of zeros and ones in an eight-bit byte (256 in fact) to accommodate all the letters and numbers in those languages, plus the special symbols and control keys. Oriental languages—and many Middle Eastern languages—require at least two eight-bit bytes to represent their written characters.

Control and Escape

Ctrl, Esc, Alt are special keys that appear on some keyboards. They send special codes to the computer. When you press one of these keys, it alerts the computer that something out of the ordinary is expected.

- Ctrl S tells the computer to stop words on the screen from scrolling upward.
- Ctrl Q tells the computer to resume scrolling.

- Esc permits you to enter special coding sequences into whatever you are typing. Embedding coding sequences into anything that you write is an advanced use of the computer. Avoid doing this unless you know what the results will be.
- Ctrl A permits someone using the console to respond to requests from others who are using the computer. It has other functions, too.
- Ctrl B permits someone using the console to perform a soft or hard reset of the computer. Do not use Ctrl B unless you are thoroughly familiar with the consequences of soft and hard resets. Misuse of either function can have serious consequences for other users. You will find information on resets in *Using Your System*.

Letters and numbers

When you enter A Return on your keyboard, the computer will know that you mean the symbol `A`. If you are entering text into a document, you can be certain that the computer will faithfully represent the letters of the alphabet precisely as you intend to use them.

Numbers, however, are different. They present the computer with the need to make a decision.

The number keys 0, 1, 2, ... , 9 send their own codes to the computer. But pressing 3 could mean that you intend to do arithmetic using the *number* 3, or it could mean that simply want the *character* 3 (a shape on the screen) to appear in your letter. The ASCII code sent by 3 is 51.

Within the computer, ASCII code 51 is associated with the character (a shape) that looks like 3. That sounds awkward.

Left to itself, the computer has no way of deciding precisely what you intended by pressing 3.

In fact, the computer does not make the decision. It leaves the decision to the program or the command that you are using when you press 3. The program or command that you are using is "aware" of the context in which you are entering characters. A text editing program such as EDIT/3000 will conclude that you mean the character (shape) 3. An accounting program, however, will conclude that you mean the number 3 and that some arithmetic operation is intended.

Why Numbers?

Files: ASCII or binary?

When you hear or read that something is in ASCII code, you will know that what is recorded is letters or words or symbols that you can read, just as you would read any printed document.

When you hear or read that something is in binary code, you will know that what is recorded is collections of ones and zeros. You cannot read binary files as you would a document.

> To say that a text file is recorded in ASCII form is a convenient way of *thinking* about text files. It is easier to imagine letters and words being recorded just as you type them at the keyboard. In truth, *all* files are recorded in binary form.
>
> When you create a text file, the computer records this fact for its own reference.
>
> When you use a program or a command to examine a text file, the computer recognizes from its own records that the file was intended for reading by people and performs a quick interpretation of the data that it finds in order to display the letters and words that you wrote.

A Practical Problem

If you were writing a program designed to put out a fire, you might write something like this:

```
program stopfire(telephone, first aid, water){what is needed}
begin                                {start the program}
  call firemen                        {a simple command}
  get people out of the house        {a simple command}
  if someone is hurt then            {a single question}
    begin                               {start special action}
       call a doctor                        {a simple command}
       administer first aid                {a simple command}
    end                                    {stop special action}
    look at the house                  {a simple command}
    while the house is burning          {a continuing question}
       begin                              {start special action}
          pour on more water                      {a simple command}
          look at the house                  {a simple command}
      end                                  {stop special action}
  end                                  {stop the program}
```

The idea behind creating programs is no more complicated than this. True, most computer programs *are* more complicated. But the idea is the same.

You might re-write the program to specify even more details about what to do.

Here, `begin` and `end` are commands—not to perform a particular action, but to show where particular actions (or groups of actions) start and stop.

Make note of `while`. It introduces a question that will be asked not once, but many times. The `begin` and `end` that follow it surround particular actions that will be performed over and over, until the house is no longer burning.

The ability of programs to control repetitive actions such as these give computers much of their power to solve problems. Programs divide big problems into small problems and speedily work at solving the small problems until all of the little solutions add up to a big solution.

Addition—Computer Problem Solving

You may recall this small problem in addition from Chapter 2.

```
234
 19
  8
611
----
872
```

Computers solve such problems by following the instructions designed into them. The instructions are written in one of the many programming languages that are available.

> There is no need to learn a programming language, unless the subject catches your interest or your job requires it. Indeed, you may wish to skip over the following example and continue your reading with the next chapter. You will lose nothing by doing so. The value of the following example lies in shedding a little light on how computers do what they do.

Here is one computer solution to the task of adding numbers. This program will take numbers from the keyboard, one after the other, and continue adding them together, one after the other, until you enter a zero (0). Then it will display the sum of the numbers that you entered. The instructions are written in a programming language called Pascal:

```
PROGRAM ADD(INPUT, OUTPUT);                {line 1}

VAR DIGITS, TOTAL : REAL;                  {line 2}
BEGIN                                      {line 3}
TOTAL := 0;                                {line 4}
readln(DIGITS);                            {line 5}
while DIGITS <> 0 do                       {line 6}
      begin                                {line 7}
      TOTAL := TOTAL + DIGITS;             {line 8}
      readln(DIGITS);                      {line 9}
      end;                                {line 10}
writeln(TOTAL : 10 : 2);                  {line 11}
END.                                      {line 12}
```

Unless you have encountered programming, it looks odd. If you have any acquaintance with programming, you will recognize that it is simple, unsophisticated, perhaps a bit primitive. It is not the only possible method—nor is it necessarily the best—but it accomplishes the task of adding together numbers. Someone else might design a different method, using different steps. Someone else might use a different programming language—perhaps the language called C, or COBOL, or FORTRAN, or yet another language.

Whatever the language, whatever the method, each method would be considered an *algorithm*, in this case, an algorithm for adding numbers. Algorithms, like the rules that you use for adding numbers, are sets of instructions—recipes, if you like—for solving a problem or accomplishing some task.

The algorithm described here does not tell us what the sum of the numbers might be. Rather, it describes, in minute detail, the steps that are to be followed in adding numbers together to produce a sum.

Each programming language has its own methods, its own "grammar," its own "punctuation," and its own vocabulary.

If you do nothing else, make note that the instructions in this program are very specific.

1. `PROGRAM ADD(INPUT, OUTPUT);` {line 1}

   This program *statement* is an introduction. It tells the computer "this is a program called ADD, and it will accept input from the user and produce output."

   In the vocabulary of Pascal, it tells the computer quite specifically that information will come from the keyboard (`INPUT`, which happens to be the file `$STDIN` during a session). And it tells the computer that information will be returned to the video screen (`OUTPUT`, which happens to be `$STDLIST` during a session). Now the computer knows which of its parts (in addition to the processor) will be active.

   A different introductory statement could be used to tell the computer that information will come from another source or go to another destination.

2. `VAR DIGITS, TOTAL : REAL;` {line 2}

   This is another introductory statement. It tells the computer "create two variables (places to hold something in memory) and make them big enough to hold numbers with fractions." Numbers without fractions take up less room in memory than do numbers with fractions.

   `VAR` means "variable." In the vocabulary of Pascal, it tells the computer that the words `DIGITS` and `TOTAL` will be used to hold something.

   `REAL` tells the computer that what will be held in `DIGITS` and `TOTAL` will be numbers and, more precisely, that these numbers might include fractional parts (263.6, for instance).

   For as long as the program is active, the computer understands that `DIGITS` and `TOTAL` really stand for numbers. Numbers without fractional parts are called integers.

   This might seem unnecessary. Computers work with numbers, do they not? But this and every other statement is vital to the program. From this statement, the computer knows that whatever is put into the variables `DIGITS` and `TOTAL` *must* be numbers (and only numbers), not letters or words. It knows, too, that it must create in its electronic memory two distinct places to hold numbers.

3. `BEGIN {line 3}`

   "Now, after all this introductory material, here is what you (the program) will do." This statement signals the end of the introductory material and the beginning of the main body of the instructions.

4. `TOTAL := 0; {line 4}`

   "Begin with a sum of zero." The computer must be told explicitly where to start. Where it starts is by putting the value 0 (zero) into the area of memory represented by the variable `TOTAL`. Without this statement, there is no way to predict what the starting value of `TOTAL` might be, and that could cause problems. (As yet, the area of memory represented by the variable `DIGITS` holds no useful value.)

5. `readln(DIGITS); {line 5}`

   "Take a number from the keyboard." Implicit in this statement is the instruction to "remember" the number that comes from the keyboard by putting it into the memory area called `DIGITS`.

6. `while DIGITS <> 0 do {line 6}`

   "Examine the number taken from the keyboard. *If* it is not equal to zero, look for the `begin` following `do` and start there. If the number is zero, proceed to the step following the next `end` that you find." (The <> means "is not equal to" or "does not equal.")

   There are a lot of instructions in this one, short statement. Programming languages are precise, and they compress a lot of meaning into very few words.

   A very natural question at this point is, "Why not say 'if' if you mean 'if'?"

   The answer lies in the meaning of the "words" used in the Pascal instructions. Pascal's `while` and `if` are called *conditionals*—they are decision-makers. In its *simplest* use, Pascal's `if` means "make this comparison, make a decision, then continue."

   Pascal's `while`, on the other hand, means "for as long as some condition is true...." And that is what we want the program to do: we want to be able to enter *many* different numbers, over and over, and have them added,

one after another, to yield a sum. And we want the computer to *stop* taking and adding numbers when we enter a zero.

In the language of Pascal, `while` is a convenient way to start a *loop*—a sequence of steps that is repeated, over and over, until we give the computer some reason to cease the repetition. Each time we enter a number, `while` says: "Is this zero? If it is not zero, `do` whatever comes after the next `begin`. If it is zero, look for the next `end` and pick up the instruction that follows."

7. `begin {line 7}`

"Start here and follow the next sequence of instructions until you find an `end;` statement." What follows are the instructions to be followed over and over (in a loop)—until zero is entered at the keyboard.

8. `TOTAL := TOTAL + DIGITS; {line 8}`

"Perform addition." Clearly this is not the addition that you were taught in school. It is, however, correct and precise in the "grammar" of Pascal.

More accurately, this instruction reads: "Take the value found in the memory area called `TOTAL`, add to it the value found in memory area called `DIGITS`, and insert the sum back into the memory area called `TOTAL`."

`TOTAL` no longer holds the value zero. Zero has been "erased." Instead it holds the new value found by adding the old value to whatever number that we entered at the keyboard.

9. `readln(DIGITS); {line 9}`

"Take another number from the keyboard and place it in the memory area called `DIGITS`." The new number that we entered takes the place of the number we entered previously.

10. `end; {line 10}`

"End of the loop." This is a form of punctuation. It concludes the steps begun in statements 6 and 7. And in this case, it sends the program running back to the `while` conditional. Because `while` signals repetition, the computer returns to the instruction in statement 6 and finds that it must perform another examination of the number entered.

If at any time we enter 0 (zero), the computer will discover that the *condition* defined in statement 6 (`is not zero`) has not been met. Until we enter zero, the computer will continue to take numbers, one number at a time, perform addition, juggle its memory, and wait for still another number.

Recall that this circular repetition of one or more steps is called a *loop*.

If we enter 0 (zero), the computer will break out of the loop dictated by `while` and proceed to the next (and in this instance, the last) step in the procedure.

11. `writeln(TOTAL : 10 : 2); {line 11}`

    "Show the total that you computed." This last instruction tells the computer to display the sum of all of the additions that it has performed. More accurately, it displays the value that it finds in the memory area called `TOTAL`, and that value is the sum of the additions. A piece of the instruction, `: 10 : 2`, tells the computer how to display the number—in this case, as a number occupying the space of ten characters on the screen, with two digits following the decimal point.

    Displaying images on the video screen is also under the control of the system processing unit.

12. `END. {line 12}`

    "The program is done." This is the last mark of punctuation. This tells the computer to stop the program.

There is one serious flaw in this program. If at any time we enter a letter, or a string of letters, or a combination of numbers and letters, instead of a number, the program will not know what to do. It has not been told what to do. The program instructions specified only numbers as input.

In the face of this confusion, the computer will bring the program to a grinding halt. The program *aborts* on the error we made—entering a letter instead of a number.

A programming engineer would include in this simple program very specific instructions about what to do if the person using the program enters something other than a number. Very likely a better-designed program would detect such

an error and display a message. Then it would suggest an answer and wait patiently for another number and continue with the additions.

> As cryptic as the Pascal language might seem, it is still too close to everyday language for the computer to use efficiently.

Computers truly understand only collections of ones and zeros. Before this or any other program is ready for the computer's use, it must be translated into the binary language that the computer understands, called *machine language*. Other programs called *compilers* perform this translation.

7

Commands

There are well over 200 commands built into the MPE/iX operating system. Some are as simple as `SHOWME`. The `SHOWME` command requires only that you enter its name at the prompt and press Return:

```
SHOWME
```

Others are more involved, and some of them are complex.

A "simple" command does one or two things and does not offer much flexibility in its execution. A "complex" command may do one thing—or many things. In addition, a complex command usually permits you to specify in great detail exactly what the command will do and how it will do it.

Some programs, such as HP Easytime/iX or EDIT/3000, automatically call upon complex commands and permit you to accomplish what you want to do without having to understand the command in all its complexity. The EDIT/3000 program creates text files for you and gives you a way to change these files as you need.

You may want to rely on HP Easytime/iX or other programs to do most of your work on the computer. In some situations, however, no program will accomplish *precisely* what you intend.

The commands built into MPE/iX provide tools to accomplish those ends with almost any degree of that control that you might wish to exercise.

Types of Commands

Not every command is available to every user. Some commands are restricted to those who have the authority (capability) to use them. A handful of commands are available only to the person using the system console (the system operator or the system administrator).

Restricted commands

The restrictions on some commands are an extension of the same capabilities that apply to groups, accounts, and users.

The `NEWUSER` command is restricted. This command allows an account manager (AM capability) to define new users (user names) who will be permitted to log on to an account.

The `NEWACCT` command creates (names) new accounts. Only users who have system manager (SM) capability have the authority to execute `NEWACCT`.

How will you know whether you have authority to use a particular command? If you try to use a command that exceeds the capabilities assigned to you, the operating system will inform you through an error message on your terminal screen. There are no penalties for experimenting.

Console commands

A small number of commands is tightly restricted. These commands are critical to the efficient management of the computer and its resources.

The `ALLOW` command permits a system operator or a system manager to grant or deny the use of other, restricted commands. Such a command has the potential for misuse, and so the system permits execution of `ALLOW` only at the system console.

Again, there are no penalties for attempting to use `ALLOW` or the other *console commands*. The system will simply inform you that such commands are reserved for execution at the console.

Command Parameters

Simple commands, such as `SHOWME` and `SHOWTIME`, need—and want—nothing more from you than this: type the name of the command and press Return.

`SHOWME`Return

`SHOWTIME`Return

The complex commands offer greater flexibility in their execution, but this flexibility comes at a price. The complex command *may require* additional pieces of information from you in order to execute successfully.

The pieces of information that you add to a command are called *parameters*. You add pieces of information after typing the command name and before pressing Return. The `PRINT` command accepts a parameter, the name of a file:

`PRINT` *filename*Return

Using italics here (*filename*) indicates that you would have to substitute the name of an existing file where *filename* appears, as in this example:

`PRINT TAXJAN`Return

> Taken together, `PRINT TAXJAN`Return make up a *command line*—a line consisting of a command, plus its parameters (if any), terminated by Return. You might think of a command line as a package of instructions for the computer.
>
> - Use the `PRINT` command.
> - Use it on the `TAXJAN` file.
>
> Return ties up the package and sends it to the computer.

A command gives instructions to the computer. In the same way, parameters give instructions to a command. Parameters control or modify the way in which the command executes.

Some commands accept only one parameter. Some accept many parameters.

Command Parameters

Parameters fall into these two groups:

- Required parameters

  A required parameter is a piece of information that you *must* add to the command in order to have the command execute.

- Optional parameters

  An optional parameter is a piece of information that you may *choose* to add to the command. Add the information if it suits your purpose.

  Be aware, though, that choosing *not* to include an optional parameter will cause the computer to do *something*. What will it do? That depends upon which command you are using. The `LISTFILE` command is an example.

Optional parameters

`LISTFILE` commands accepts several parameters, but all of them are optional.

```
LISTFILE
```
Return

Used without any of its optional parameters, `LISTFILE` displays the names of all of the files in the group where you are logged on. That is the action that this command takes when you do not specify any of its parameters.

Suppose, instead, that you want to see only the names of those files that begin with the letter "G." If that is what you want, add one parameter to the `LISTFILE` command line:

```
LISTFILE G@
```
Return

Wildcard Characters: @ ? #

Character	Meaning
@	This tells the computer to look for any combination of *letters or numbers* (none or as many as 8). In this case `G@` tells the computer to look for any file name in your logon group that begins with the letter `G`
?	This is the equivalent of "any single character, *letter or number*." `G??` tells the computer to look for any file name beginning with the letter "G" and having (at most) two more letters or numbers in the name.
#	This is the equivalent of "any single *number* (digit) 0 through 9."

> `@#?` translates to "any name beginning with any combination of letters or numbers that ends in a single number followed by a single letter or digit."
>
> Files called `FIN2A`, `ACCT33`, and `TAXRPT2G` would fit the specification described by `@#?`.
>
> Files called `FINAL2` or `ACCTS3` would not fit the specification.

Adding the parameter `G@` permits you to narrow the search for file names to the ones you wanted—file names beginning with "G."

Recall that `LISTFILE`Return by itself gives you all of the file names in your logon group. Why does it do that?

The programming engineers who created the `LISTFILE` command knew that someone using `LISTFILE` would want to see the names of files, but which files? They had no way of knowing what you might choose, so they made two decisions.

The first decision was to leave the choice to you—the parameter for specifying files is optional. You choose. The second decision concerned what `LISTFILE` should display if you make no choice at all.

This second decision involved creating a *default*—a set of instructions for `LISTFILE` to follow if the user does not specify any parameters. By default, `LISTFILE` lists all the file names it can find in your current group if you decide *not* to narrow the search. With this release, `LISTFILE` contains additional parameters. For more information, refer to the book, *New Features of MPE/iX: Using the Hierarchical File System* (32650-90351).

Every optional parameter of a command has some default action. You will find information about optional parameters and their defaults in book *Commands Reference - HP 300 Series 9X8LX* (B3813-90011), and under the **Parameters** heading, of each command in the MPE/iX Help Facility.

Required parameters

In contrast to optional parameters, required parameters are pieces of information that you *must* give to a command in order to have the command execute successfully.

The command to remove or erase a file, `PURGE`, requires that you supply at least one parameter, the name of the file to be removed:

```
PURGE ACCTRPT[Return]
```

This use of `PURGE` removes the file called `ACCTRPT` from the group in your logon group. For `PURGE`, the name of *some* file is a *required parameter*.

If you accidentally omit the name of a file to be purged, the operating system will display an error message informing you of your mistake. This is true of any required parameter of any command. Omitting required information from the command line generates an error message.

How do I find out which parameters go with each and every command? Two sources of information are readily available:

- *Commands Reference - HP 3000 Series 9X8LX* (B3813-90011)
- the MPE/iX Help Facility

  The *MPE/iX Commands Reference Manual, Vols I and II* (32650-90003), a comprehensive manual that includes details on all MPE/iX commands.

Help with Commands

If you look at the PURGE command in the book *Command Reference - HP 3000 Series 9X8LX* (B3820-90007), you will find, under the heading **Syntax**, a curious-looking entry:

PURGE *filereference*[,TEMP]

This method of showing a command and its parameters is called a *syntax diagram*.

Filereference looks unfamiliar. It is simply the *formal name* of the (required) parameter for specifying a file to erase.

> A formal name is not a parameter. Formal names such as *filereference* appear in syntax diagrams to help you understand what kind of information belongs to a particular parameter.
>
> The information that you give to PURGE, or any other command, when you execute it, is the *actual parameter*.

On the command line, you would enter a suitable value (a name, a word, a number) to achieve the results that you want, just as you enter PURGE ACCTRPT(Return) to remove a file called ACCTRPT. In this case, the information ACCTRPT is the actual parameter.

Command	Formal Parameter A Name To Help You	Actual Parameter A Real File to Erase
PURGE	*filereference*	ACCTRPT
Syntax Diagram	PURGE *filereference*	
Command Entered	PURGE ACCTRPT	

Filereference is a required parameter. It follows the name of the command, and there are no square brackets around it. *Filereference* suggests that some reference to a file is called for. In this case, it calls for the name of a file.

You may find `,TEMP` unfamiliar, too. It is an optional parameter. It is also a *keyword*. There is more on information on keywords in "Variables and keywords" and in "Syntax diagrams".

In the syntax diagram, the square brackets (`[ ]`) surrounding `,TEMP` signify that using `,TEMP` is optional. You may use it or omit it according to your need. If you choose to use it, you must include the comma that precedes `TEMP`, as in this example:

```
PURGE K0356774,TEMP
```

This command line removes a *temporary* file named `K0356774` from the disk. If no such temporary file exists, the operating system displays an warning message telling you that it could not find that (temporary) file.

Syntax diagrams

Syntax diagrams help in understanding the parts of a command—its parameters—their use, and their relationship to each other. When the command is a simple one, its syntax diagram is relatively easy to follow. But when the command is complex, when it has many parameters, its syntax diagram does appear formidable.

But, despite any *potential* complexity, all syntax diagrams follow simple rules.

What follows is not a complete lesson in reading syntax diagrams. Instead, it is an introduction intended to make syntax diagrams look a little less formidable.

Variables and keywords

The syntax diagram for the `PURGE` command is simple.

`PURGE` *filereference*[`,TEMP`]

The parameter *filereference* is in italic lettering to indicate that it is a *variable*. Entering `PURGE filereference`Return would produce an error. Because *filereference* is in italics, you must use the *name* of a file in its place when you execute the `PURGE` command:

```
PURGE ACCTRPT(Return)
```

Variable—required parameter

What is a variable? You might think of it as an empty box. Until you put something inside it, the box is just an empty box. When you put something inside it, it becomes a box of.... What did you put inside the box?

For a computer, a variable is much like an empty box. It is an area of memory waiting for something to fill it. In a syntax diagram, a formal name such as *filereference* represents just such an empty box. But *filereference* is also a parameter of the command PURGE.

For PURGE, the parameter *filereference* is a variable. That means that when you execute PURGE, it will look for something in a memory box called *filereference*.

If it finds the name of an existing file, it erases that file.

You put the name of a file into that memory box when you execute PURGE this way:

```
PURGE ACCTRPT
```

Since *filereference* is a required parameter of PURGE, the command *expects* to find something in the *filereference* memory box. If it finds nothing (because you left out the name of a file), it reports an error to the computer and the computer puts an error message on your video screen.

```
PURGE 6(Return)
```

will not work. 6 is not the name of a file, but a number.

```
PURGE X6
```

will work, *if* X6 is the name of a file in your current group *and if* the file is not protected in some way.

> *Worth Knowing*
>
> In the Help Facility, variables appear not in italic lettering, but in plain lowercase letters.

Variable—optional parameter

Optional parameters are variables, too, but with a difference. The `LISTFILE` command serves as an example. Its syntax diagram includes this parameter:

`LISTFILE` [*filename*]

Here, *filename* is an optional parameter.

There is a memory box called *filename*, too. But for `LISTFILE` that box always has something in it. What is there is a value that says: `all files in the group`. And that is what `LISTFILE` displays if you do not specify a file name. Unlike `PURGE` with its required parameter, `LISTFILE` can be quite happy even if you do not specify a file name (because there is always something for it to find in the memory box called *filename*).

```
LISTFILE ACCTRPT
```

When you specify a file name in this fashion, the file `ACCTRPT` goes into the memory box called *filename*. When `LISTFILE` peeks into that memory box, it finds—not "all files in the group," but the name of one particular file, `ACCTRPT`.

- A required parameter (variable) is a memory box that that starts out empty. It must have something put into it. And you must put that something in it.
- An optional parameter (variable) is a memory box that starts out with a default value already in it. You can choose to put something else into the memory box, or you can choose to accept what is already there. If you choose to accept what is already there, the computer will take some default action. By default, the command will act on whatever it finds in the memory box (or boxes) that belong to it.

Under the **Parameters** heading of the `PURGE` command in the *Command Reference - HP 3000 Series 9X8LX* (B3820-90007), you will find a description of each parameter of the `PURGE` command. That description tells you what is appropriate to use.

Keyword—not variable

Look again at the syntax diagram for PURGE.

PURGE *filereference*[,TEMP]

,TEMP is also a parameter. It happens to be optional, too, because it is surrounded by brackets. But it is not a variable. It is a *keyword*. It appears in uppercase letters. It instructs PURGE to remove a *temporary file*, rather than a permanent one.

A keyword is not a variable. It is not an empty box into which you can put what you like. In order to use a keyword, you must enter the keyword exactly as it appears in the syntax diagram. You may enter the word in uppercase letters or in lowercase letters. Entering this:

PURGE ACCTRPT,TIME(Return)

will produce an error. In fact, entering *anything* other than ,TEMP after the name of a file will produce an error.

Because TEMP is optional, you do not have to use it when you enter the command. But if you do decide to use it (in order to erase a temporary file), TEMP is the only value that PURGE can accept.

The elements of syntax

There are valuable "clues" in any syntax diagram:

Syntax Elements

Element	Name	Meaning
[ ]	Brackets	What is inside is optional. Pay attention to the punctuation, the comma or the semicolon if there is one.
{ }	Braces	What is inside is required. Pay attention to any square brackets that may surround the braces, as in this construction: `SOMECOMMAND` *alpha* [; { *beta* / *gamma* / *delta* }] In the MPE/iX Help Facility, it might look like this: `SOMECOMMAND ;alpha` [; { `beta` / `gamma` / `delta` }] This construction shows up to alert you that choices are available. But it tells you that your choices are restricted. Here, you may choose whether to use *beta*, *gamma*, or *delta* or simply ignore them all. But, if you choose to use one of them, you must use *beta* or *gamma* or *delta*—but only one of them, not two or three.

Syntax Elements (continued)

Element	Name	Meaning
lowercase lowercase	Lowercase Letters	What is shown in lowercase letters is a variable. In your documents, variables appear in *italic* lowercase lettering. In the Help Facility, variables appear in plain lowercase lettering. If it is an optional parameter, you may give the command some information in place of that variable when you type the command line. If it is a required parameter, you must give the command some information in place of that variable when you type the command line. For `PURGE`, a file name is required. For `LISTFILE`, a file name is optional.
UPPER-CASE	Uppercase Letters	What is shown in uppercase letters is a keyword. If you are going to use a keyword (or something "attached" to a keyword by an equal sign =), you must enter the keyword exactly as it is spelled. What is attached to a keyword by an equal sign is usually a *variable*.
, and ;	Comma, Semicolon	Separators or place-holders. Commas and semicolons tell the computer where one parameter ends and another begins. If you omit an optional parameter that has punctuation, be sure to enter the comma or semicolon on the command line; otherwise, the computer is likely to become lost or confused. If it becomes lost or confused, it will show you an error message or—worse—do something unexpected.

Defaulted or Omitted Parameters

When you enter a command line, the operating system examines everything that you typed up to the point at which you pressed Return. If you have not mistyped the name of the command, or any of its parameters, the operating system searches its memory concerning the command that you entered.

The operating system executes a series of tests to determine how to execute the command:

- If the command is `PURGE`, look up the parameters of `PURGE`, *and*
  - If a file is named, look for the file, *and*
    - If the file exists, determine whether `,TEMP` is specified, *and*
      - If `,TEMP` is specified, then erase a temporary file; *otherwise*, erase a permanent file

This description is an over-simplification, but it illustrates the kind of "reasoning" that the computer follows when it tries to execute a command that you enter.

For every parameter of every command, there is an *otherwise* that tells the computer what to do *if* the following occurs:

- The user decides to ignore an optional parameter.
- The user fails to enter a required parameter.
- The user simply mistypes the command or one of its parameters.

In this case, the *otherwise* of `,TEMP` is: "erase a permanent file, if it can be found."

The *otherwise* (default) for a *required* parameter that you fail to specify, that you mistype, or that the computer cannot, for some reason, accept is to display an error message.

```
:PURGE
       ^
REQUIRES FILE NAME TO BE PURGED.  (CIERR 381)
```

`CIERR 381` identifies the error that the system encountered. It was a CI (command interpreter) error, an error made when the command interpreter attempted to carry out this instruction. It was error number 381 in the catalog of errors that the system recognizes and reports.

PRINT—A Closer Look

The syntax diagram for the `PRINT` command is a little more complex than the ones illustrated so far. It looks like this:

`PRINT`[[`FILE=`]*filename*] [; [`OUT=`] *outfile*]

> This is only part of the syntax diagram for `PRINT`. It has other parameters, in addition to the ones shown here

The *filename* parameter is optional.

Notice, though, that you have the choice of whether to precede that parameter with `FILE=`. (Within MPE/iX, there are historical and engineering reasons for making this option available.)

The parameter *outfile* is optional, and so is the decision whether to precede it with `OUT=`. Look carefully: the semicolon is *required* if you choose to specify an *outfile*.

What is an *outfile*? It is an optional parameter. It permits you to specify the destination for the contents of *filename*. Because it is an optional parameter, *outfile* has a default action—in this case, a destination for the contents of *filename* if you decide *not* to specify a destination.

The default destination is your video screenl. From the computer's point of view, your terminal is a file, too.

Suppose that you tried this:

`PRINT TAXJAN;XYZ`Return

What happens?

You *have* specified an *outfile*, so the contents of the file will *not* go to the video terminal.

In this situation, the computer obligingly creates a *temporary* file called `XYZ`. You could find it with `LISTFILE @;TEMP`Return. This file will disappear as soon as you log off.

You could transform this temporary file into a permanent file by using the SAVE command:

```
SAVE XYZ(Return)
```

Worth Knowing

PURGE removes only one file at a time: PURGE G@(Return) will not remove all of the files beginning with the letter "G." It will, instead, generate a message informing you that you made an error. You must use PURGE each time that you wish to remove a single file. That helps to ensure that you make a conscious decision to erase a particular file.

MPE/iX has no command to "un-erase" a file. Backing up files is vital. You can bring back a file from tape by using the RESTORE command if the file was copied to tape with the STORE command, but that does not help if the file that you erased has not been recorded on tape.

Regular and frequent backups give you the best protection against the inadvertent erasure of a file.

8

Command Files and Jobs

During a session, the computer accepts only one command line at a time. But over the course of days or weeks or months, you are likely to identify groups of commands that are valuable and change very little. This is especially true if you have the responsibility of managing your computer system.

There is a way to execute this group of commands all at once, instead of having to enter each command one at a time every time. In fact, there is more than one way.

- One way is to create and use *command files*.
- Another way is to create and use *job files*.

Command files and job files are files that you create using a text editor of some kind, such as EDIT/3000. The only requirements are that the editor that you use must produce ASCII text and that each line of text contain no more than 72 characters. In a sense, each file is a small program that you write, using the commands and parameters that have become familiar and useful to you.

Creating command files or job files is far easier than programming. By the time you find yourself asking "Is there some way to ... ," you will have almost all of the knowledge that you need to create these files. However, each kind of file obeys its own rules, and each operates a little differently.

Command Files

Command files are the easiest to create.

The file that you create should contain the commands that you want to execute, plus any parameters that you want to supply to the commands. You may put into each file as few, or as many, commands as you need.

When you save the file to disk, give the file a name that has some meaning to you, something that you will remember.

To execute the files, all that you need to do is type its name at the keyboard and press Return.

You will find instructions on creating command files in the book *Task Reference - HP 3000 Series 9X8LX* (B3820-90005).

Jobs and Job Files

Like command files, job files free you from having to perform the same sequence of tasks over and over again.

You might think of a running a job as a little like delegating your work to someone else who goes off into another room and works quietly, without supervision, until the work is finished. If you are confident about the work that this other person will do—about the job that you design—the savings in time and effort on your part can be considerable.

At the same time, jobs offer advantages—and disadvantages—over using a command file.

- The first advantage of using jobs is that they can save you time when you need to perform a series of tasks that is complex or time-consuming.
- The second advantage of using jobs is that you can instruct the computer to execute the job at a later time, perhaps after you have gone home for the day.
- The chief disadvantage of using jobs is that it takes a little more time and effort to create the job.

When an error—or something unexpected—occurs in a job, it *can* take a little time to determine exactly why something went amiss.

Job or Session?

A major difference between a session and a job is the difference between "right now" and "sometime in the future."

During a session, the computer attempts to execute your instructions as soon as it possibly can.

A job, however, executes your instructions in one of these two ways:

- at some future time that you specify
- as soon as the computer has sufficient time and resources available

You can tell the computer when to begin executing a job—or you can let the computer decide the earliest opportunity for executing your job.

There are other differences between jobs and sessions.

- The `SHOWJOB` command displays sessions as `#Snnnn`, where `nnnn` is the session number.
- Sessions are *interactive*. You start an interactive session by logging on with `HELLO` and continue your dialogue with the computer, entering commands or running programs as you need them.
- The `SHOWJOB` command displays jobs as `#Jnnnn`, where `nnnn` is the job number.
- Jobs are not interactive. You start a job *during a session* by entering the `STREAM` command, followed by the name of a job file that contains a list of commands. Then you continue with your session, entering commands or running programs as you need them—or perhaps you log off and go home for the evening.

During a session, your terminal becomes unavailable to you for the fraction of a second that it takes for the command to finish executing. As soon as the execution terminates, the computer shows you another prompt and waits for your next instruction.

Job or Session?

However, there are commands and some special programs that require considerable time to finish executing. During the time that a command or program is executing, your terminal is unavailable to you. It could be unavailable for seconds, or even for hours. That can happen during some complex system administration tasks, such as backing up your files. It can happen, too, if the computer is using all of its resources to satisfy the needs of many users.

At such times, it is only natural to want for a way of recapturing the use of your terminal, or to defer the execution of some task until the computer is able to devote a larger portion of its resources to your work.

Jobs answer both requirements by running *in the background*, more or less out of sight. Indeed, when it does execute, your job will claim none of your time, nor will it "steal" your terminal from you. While a job is running, your terminal is free for whatever other work you might want to do. You can even tell a job when to run: hours, days, weeks, months, even as much as a year from now. You do not have to be at your terminal while a job runs.

There is a drawback, however. A session is called "interactive" because you and the computer are engaged in a "dialogue"—a conversation of sorts—during which you can issue any valid command and see the results almost immediately. You are free to change your mind, modify a command, and try it again. If the command or program that you are running prompts you to provide more information, you can enter the correct response then and there. Errors that occur give rise to error messages on your video terminal and permit you to analyze them and correct any mistakes.

Jobs, by contrast, are *not* interactive. When you send a job to the computer for processing, your ability to influence the job is usually confined to suspending the execution of the job (`BREAKJOB` *jobnumber*), resuming its execution (`RESUMEJOB` *jobnumber*), or terminating the job (`ABORTJOB` *jobnumber*).

> There are techniques for designing a job in such a way that the job will stop to prompt you for more information. A job of that sort, however, requires your presence at the terminal.

Starting jobs through HP Easytime/iX

Starting a job through HP Easytime/iX is accomplished through the Job Management screen. You will find information about starting jobs through HP Easytime/iX in the book *Getting Started - HP 3000 Series 9X8LX* (B3820-90003).

Starting jobs through MPE/iX Commands

Starting a job through MPE/iX commands involves using the `STREAM` command. For that reason, running a job is sometimes called "streaming a job." A loose analogy might be a pipe through which jobs "flow" on their way to some processing. At any time, there might be one or many jobs in the "pipe" on their way to processing.

The simplest way of starting a job is to enter something like this:

```
STREAM TAXJOB
```
Return

`TAXJOB` is some job file that you have created and saved to disk.

Do not confuse the `STREAM` command, which any user can execute, with the `STREAMS` command, which is available only to a user with SM or OP capability.

- `STREAM` sends a job to the computer for execution (puts it in the pipeline).
- `STREAMS` permits an operator or a manager to "open or close the pipe" for streaming jobs.

You cannot use the `STREAM` command until someone has executed the `STREAMS` command to permit the streaming of jobs.

You will find more details on the mechanics of starting a job in the book *Task Reference - HP 3000 Series 9X8LX* (B3813-90009).

When You Run a Job

This last section on jobs is devoted to points of interest, rather than to the steps involved in creating and starting jobs. *Using Your System* includes several chapters that guide you through the creation of job files and starting jobs.

At the start

You may leave the decision of when to execute your job to the computer. Or, within limits, you may increase the priority of your job in the *job queue* in order to have it execute sooner.

Alternatively, you may specify a time (a time of day, a day of the week, a week of the month, even another year) for your job's execution. And at that time, if the computer has resources available, it will execute your job.

The command to start a job executes almost immediately, and, just as quickly, the computer returns control of your terminal to you.

Before returning control of your terminal to you, the computer will display the *job identification number* assigned to your job. It might be `J345`, meaning that yours is job number 345 in a sequence of jobs that were launched since the job counter was reset.

This done, you may continue with other work. You have only to wait for the job to execute and finish and then examine the *job listing* on your printer.

Worth Remembering

Record the job identification number of your job. While the job is running, this number is your link with the job. You may want to do several things that are possible if you know the job number. Among them are:

- monitor the job status (`SHOWJOB`)
- suspend the job (`BREAKJOB`)
- resume the job (after suspending it) `RESUMEJOB`
- terminate the job (`ABORTJOB`)
- obtain status reports on any spool files associated with your job (`LISTSPF`)

These and other commands, along with the `JOB` command, are discussed in greater detail in *Using Your System* and in the MPE/iX Help Facility.

Your job—your list of commands for execution—is transformed into an input spool file and, like all other input spool files, waits its turn to execute. It may wait a few seconds, a few minutes, or even a few hours before executing. In the meantime, you may continue with other work on your terminal.

Job priority

Jobs are executed in the order of their priority, and priorities range from 1 (the lowest) to 13, with `HIPRI` (the highest) reserved for your system manager or system operator. If you do not specify a priority when you create your job file, your job will be assigned a priority of 8, which is just above the halfway point in the range of priorities.

If two jobs have equal priority, the job that was started earlier takes precedence. Here, "started" means that a job was sent to the computer for processing by using the `STREAM` command.

Getting over the fence

Your system manager or system operator will use the `JOBFENCE` command to set the priority limits for jobs. If the *jobfence* is set at 7 and your job has a priority of 8, it will eventually execute, but only after jobs having a priority of 9 or greater have executed. If your job has a priority equal to or less than 7 (the same as the jobfence), it will not run.

If the jobfence is set at `HIPRI`, no jobs except those given `HIPRI` priority will execute. Only a system administrator or a system operator is able to assign `HIPRI` to a job.

`JOBFENCE` gives your system manager or system operator a means of controlling the flow of job traffic on your system.

When you create a job, you may use the `JOB` command and its `;INPRI` keyword to increase the input priority of your job, even before you launch it. Decreasing the input priority to 1 insures that the job will never run—occasionally useful if you want to put a job "on hold."

You may use the `SPOOLF` command with the `;ALTER` and `;PRI=` parameters to increase (or decrease) the output priority of the spool files that are generated by your job.

> It is tempting to give your jobs (or output spool files) the highest possible priority and race to the head of the line.
>
> Some jobs truly do deserve higher priority—but only some.

Scheduling

The `STREAM` command includes the keywords `;AT`, `;DAY`, `;DATE`, and `;IN`. These keywords permit you to set the start time for a job at a particular *time*, on a particular *day*, on a particular *date*, or *in* a number of days, hours, or minutes (from now).

Errors and your job listing

If there are errors in your job file—or if the computer cannot complete an instruction in the job file—the computer will display on your video terminal nothing more than a message telling you that your job terminated in an error.

During a session, the computer displays error messages on your terminal (`$STDLIST` for a session). During a job, however, `$STDLIST` is *redirected* to the printer connected to your MPE/iX system, and that is where your error listing appears.

This redirection of information is, in fact, quite sensible. Your computer has no way of knowing whether you will be at your terminal when your job executes, whether your terminal will even be turned on, or whether you will have gone home for the evening.

To find the source of an error in your job, you must go to your printer and examine the *job listing*. The *job listing* is the printed report that your job delivers whether the job completed successfully or failed at some step along the way. It gives you confirmation of each command in the job, telling you whether the command executed successfully or failed—and if it failed, what error caused the failure.

Index

Special characters

#
  wildcards, 7-4
?
  wildcards, 7-4
@
  wildcards, 2-12, 7-4

A

`ABORT` command, 2-13
access control definitions
  ACDs, 4-10
accounts, 2-10, 3-7
  capabilities, 4-3
  manager, 4-5
  managers, 2-19
  names, 2-3, 2-15, 5-6
  passwords, 2-9
  protecting, 2-15
  structures, 2-3, 3-7
ACDs
  access control definitions, 4-10
  security features, 4-10
action
  default, 7-16
addresses
  files, 3-7
AI, 5-4
`ALLOW` command, 7-2
alphabet
  computer language, 6-3
AM
  capabilities, 4-5, 7-2
applications
  what are, 1-6
archiving
  information, 5-10
artificial intelligence, 5-4
ASCII
  codes, 6-6
  files, 3-8, 3-9, 8-1

B

BA
  capabilities, 4-7
background
  jobs, 8-4
  programs, 5-7
backing up
  full, 5-11
  information, 5-10
  partial, 5-11
  scheduling, 5-11
  system, 3-19
backreferencing, 3-16, 7-16
batch access
  capabilities, 4-7
battery backup, 1-4
binary
  files, 3-8, 3-9
  notation, 3-3, 6-4
bit
  computer languages, 6-5
braces

syntax, 7-12
brackets
syntax, 7-8, 7-12
Break
keys, 2-13
byte
computer languages, 6-5
byte stream files, 3-8

C

capabilities, 4-2, 4-3, 4-11
accounts, 4-3
AM, 4-5, 7-2
BA, 4-7
batch access, 4-7
commands, 4-3, 7-2
groups, 4-3, 4-6
IA, 4-7
interactive access, 4-7
ND, 4-7
nonshareable devices, 4-7
OP, 4-4, 7-2
PH, 4-9
PM, 4-9
privileged mode, 4-9
processes, 4-3
process handling, 4-9
programs, 4-3
save files, 4-7
SF, 4-7
SM, 4-4, 7-2
users, 4-3
central processing unit, 1-2, 5-6
characters
wildcards, 7-4
CI
programs, 5-5
CI syntax, 3-5
codes
ASCII, 6-6
control, 6-6
files, 3-9
Morse, 6-3
command
files, 8-1, 8-2
interpreter, 1-5
line, 7-3
command interpreter, 5-5
commands
ABORT, 2-13
ABORTJOB, 8-7
ALLOW, 7-2
automating, 8-1
BREAKJOB, 8-7
built-in, 7-1
capabilities, 4-3, 7-2
complex, 7-1
console, 7-2
CONSOLE, 3-13
data, end of, 3-18
:EOD, 3-18
FILE, 3-16
files, 4-11
HELLO, 2-2
JOB, 8-8
JOBFENCE, 8-8
keywords, 7-11
line, 7-14
LISTFILE, 2-12, 7-4, 7-16
LISTSPF, 5-14, 5-15, 8-7
NEWACCT, 7-2
NEWUSER, 7-2
parameters, 7-3
PASSWORD, 2-16
PRINT, 3-15, 3-16, 7-11
punctuation, 7-13
punctuation in, 5-5
PURGE , 7-6, 7-7, 7-8
RESTORE, 5-10
restricted, 7-2
RESUME, 2-13
resuming, 2-13

RESUMINGJOB, 8-7
RUN, 4-13, 4-16
SAVE, 7-16
SHOWJOB, 8-7
SHOWME, 3-11, 4-16, 4-17, 7-1, 7-3
SHOWTIME, 7-3
simple, 7-1
SPOOLF, 5-14
starting, RUN, 4-13
starting, without RUN, 4-14
starting, XEQ, 4-17
STORE, 3-18, 5-10
STREAM, 8-5, 8-7, 8-8
STREAMS, 8-5
suspending, 2-13
syntax, 7-7
SYSGEN, 3-14, 3-19
terminating, 2-13
variables, 7-10, 7-11, 7-13
XEQ, 4-13, 4-16, 4-17
commas
syntax, 7-13
compiled programs, 6-16
component, 3-5
computer languages
alphabet, 6-3
bit, 6-5
byte, 6-5
number, 6-3, 6-6
Pascal, 6-11
what are, 6-2
computers
keyboard codes, 5-2, 6-2, 6-6
power on/off, 1-2, 5-8
what are, 1-1
configuration
devices, 3-19
dialog, 3-19
files, 3-19
console
commands, 7-2
devices, 3-13
moving, 3-13
terminals, 3-13
CONSOLE command, 3-13
control
codes, 6-6
keys, 6-6
CPU
hardware, 1-2
Ctrl
keys, 6-6

D

data
files, 3-9
databases
files, 3-8
data, end of
commands, 3-18
decimal
notation, 6-4
default
action, 7-16
destinations, 3-15
logging on, 2-16
parameters, 7-5, 7-14, 7-15
destinations
default, 3-15
disk, 5-9
files, 3-10
information, 3-14, 3-16, 5-7
memory, 5-7
printers, 5-12
RAM, 5-7
ROM, 5-7
tape, 5-10
devices
class names, 3-16, 3-17, 3-18
configuration, 3-19
console, 1-3, 3-13
disk drives, 1-4

disks, 3-10
driver, 3-10
keyboards, 1-3
ldev, 3-12, 3-13
logical, 3-12, 3-13
nonshareable, 4-7, 5-13, 5-15
peripheral, 1-3, 3-10, 5-12
printers, 1-4, 3-10, 4-7, 5-12, 5-13, 7-16
tape drives, 1-4, 3-13
tapes, 3-18, 4-7
terminals, 1-3
UPS, 1-4
video screens, 1-3
diagrams
syntax, 7-7, 7-8, 7-10, 7-12
dialogs
configuration, 3-19
terminals, 5-2
disk drives
devices, 1-4
disks
devices, 3-10
files, 2-10
memory, 3-2, 5-9, 5-10, 5-11
segments, 3-2
tracks, 3-3
drives
disks, 1-4
tapes, 1-4

E

EDIT/3000
program, 3-9
programs, 5-9, 8-2
SAVE command, 5-9
editors
text, 8-1
electronic memory, 6-12
end of data, 3-18
EOD command, 3-18
equations
files, 3-14, 3-16, 3-18, 7-16
errors
messages, 5-3
Esc
keys, 6-6
executable
files, 3-9

F

features
security (ACDs), 4-10
fence
jobs, 8-8
FILE command, 3-16
file names
rules for, 3-4
files
addresses, 3-7
ASCII, 3-8, 3-9, 8-1
binary, 3-8, 3-9
byte stream, 3-8
codes, 3-9
command, 4-11
commands, 8-2
configuration, 3-19
creator, 2-13
data, 3-9
databases, 3-8
destinations, 3-10
disks, 2-10
equations, 3-14, 3-16, 3-18, 7-16
executable, 3-9
finding, 2-12
fully qualified name, 2-10
graphics, 3-8
identities, 3-3
jobs, 8-1
location, 3-4
locations, 3-7
mnemonics, 3-9

names, 3-3, 3-4
names, fully qualified, 2-10, 2-11
of commands, 8-1
owner, 2-13
permanent, 3-3
print files, 3-20, 5-13, 5-14
protecting, 5-11
segments, 3-4
source, 3-10
spool files, 3-20, 5-13, 5-14, 5-15
standard input, 3-11
standard output, 3-11
$STDIN, 3-11, 3-12
$STDLIST, 3-11, 3-12, 8-9
structures, 3-9
system, 3-19
system-defined, 3-11
temporary, 3-3, 7-8, 7-11, 7-16
what are, 3-1
file system
hierarchical, 2-3
finding
files, 2-12
fractional numbers, 6-12
full backup, 5-11
fully qualified files name, 2-10, 2-11
function keys, 6-2

G

grammar
programs, 6-11
graphics
files, 3-8
group IDs, 4-10
groups, 2-10, 3-7
capabilities, 4-3, 4-6
home, 2-16
names, 2-3, 2-15, 5-6
passwords, 2-9
protecting, 2-15
PUB, 2-17
shared, 2-17

H

hardware
what is, 1-2
HELLO command, 2-2
HFS syntax
summary, 3-5
hierarchical
file system, 2-3
home groups, 2-16

I

IA
capabilities, 4-7
identities
files, 3-3
IDs
group, 4-10
user, 4-10
implied run, 4-14, 4-16
information
archiving, 5-10
backing up, 5-10
destinations, 3-14, 3-16, 5-7
redirecting, 3-14, 3-16, 8-9
restoring, 5-10
storing, 5-10
input
I/O, 1-12
programs, 6-12
spool files, 5-15, 8-7
terminals, 3-11
what is, 1-12, 5-2
intelligence
artificial, 5-4
interactive access
capabilities, 4-7
interactive sessions, 8-4
I/O
input, 1-12

output, 1-12
writing, 1-12
italics
syntax, 7-12

J

JOB command, 8-8
JOBFENCE command, 8-8
jobs, 4-11
background, 8-4
fence, 8-8
files, 8-1
identification number, 8-6, 8-7
listing, 8-9
monitoring status, 8-7
not interactive, 8-4
priorities, 8-7
queue, 8-5
resuming, 8-7
scheduling, 8-8
starting, 8-5
suspending, 8-7
terminating, 8-7

K

keyboard codes
computers, 5-2, 6-6
keys
Break, 2-13
control, 6-6
Ctrl, 6-6
Esc, 6-6
function, 6-2
Return, 6-6
keywords
command, 7-11

L

languages
machine, 6-16
programs, 6-11
laser printers, 5-12
ldev
devices, 3-12, 3-13
line printers, 5-12
LISTFILE command, 2-12, 7-4, 7-16
LISTFILE command
options, 2-13
listing
jobs, 8-9
LISTSPF command, 5-14, 5-15
loading
programs, 5-7
location
files, 3-4, 3-7
logging on, 2-8
attempts, 2-9
default, 2-16
prompt, 5-2
without naming a group, 2-16
lowercase
syntax, 7-9, 7-13

M

machine languages, 6-16
magnitude
numbers, 6-4
managers
accounts, 2-19, 4-5
system, 3-13, 4-4
memory
active, 5-7
areas, 6-13, 6-14, 6-15
disks, 3-2, 5-9, 5-10, 5-11
electronic, 6-12
permanent, 5-10
RAM, 5-8
random access, 5-8
read only, 5-9
ROM, 5-9
messages
errors, 5-3, 7-15

mnemonics, 3-9
Morse code, 6-3
MPE/iX, operating system, 1-5
MPE syntax
  summary, 3-5

N

names
  accounts, 2-3, 2-15, 5-6
  device classes, 3-16, 3-17, 3-18
  files, 3-3, 3-4
  fully qualified, 2-10
  groups, 2-3, 2-15, 5-6
  programs, 4-16
  rules, 3-6
  sessions, 2-14, 5-6
  users, 2-3, 2-13, 2-15, 2-16, 5-6
ND
  capabilities, 4-7
`NEWACCT` command, 7-2
`NEWUSER` command, 7-2
nonshareable devices, 5-13
  capabilities, 4-7
notations
  binary, 3-3, 6-4
  decimal, 6-4
numbers
  computer language, 6-3, 6-6
  fractional, 6-12
  job identification, 8-6, 8-7
  logical devices, 3-12
  magnitude, 6-4
  sessions, 2-14
  whole, 6-12

O

OP
  capabilities, 4-4, 7-2
operating system
  programs, 1-5, 5-6, 5-7
optional parameters, 7-4, 7-8, 7-10, 7-16
options
  LISTFILE command, 2-13
output
  programs, 6-12
  spool files, 5-14
  terminals, 3-11
  what is, 1-12
output I/O, 1-12

P

page printers, 5-12
parameters
  commands, 7-3
  default, 7-5, 7-14, 7-15
  optional, 7-4, 7-8, 7-10, 7-16
  required, 7-4, 7-6, 7-7, 7-9, 7-10, 7-15
  what are, 7-3
parser, 5-5, 5-6
  programs, 5-5
partial backup, 5-11
Pascal
  computer languages, 6-11
`PASSWORD` command, 2-16
passwords, 2-15, 4-2
  accounts, 2-9
  groups, 2-9
  shared group, 2-17
  users, 2-9, 2-16
paths, 4-15
peripheral devices, 1-3, 3-10, 5-12
permanent files, 3-3
PH
  capabilities, 4-9
place holders
  syntax, 7-13
PM
  capabilities, 4-9
power
  on/off, 5-8
`PRINT` command, 3-15, 3-16, 7-11
printers

devices, 1-4, 3-10, 5-12
laser, 5-12
line, 5-12
page, 5-12
print files, 3-20, 5-13
priorities
jobs, 8-7
search, 4-16
privileged mode
capability, 4-9
processes
capabilities, 4-3
programs, 3-10
processes handling
capability, 4-9
processing unit
central, 5-6
system, 5-6
program
main body, 6-13
programs
abort on error, 6-15
background, 5-7
capabilities, 4-3
CI, 5-5
command interpreter, 1-5, 5-5
compiled, 6-16
condition, 6-14
EDIT/3000, 3-9, 5-9, 8-1, 8-2
example, 6-10
execution time, 8-3
grammar of, 6-11
input, 6-12
languages, 6-11
loading, 5-7
loop, 6-15
names, 4-16
operating system, 1-5, 5-6, 5-7
output, 6-12
overhead, 5-7
parser, 5-5
processes, 3-10
protect, 4-2
rules, 1-9
scanner, 5-5
spreadsheets, 1-9
starting, 4-13, 4-14, 4-17
statement, 6-12
variables, 6-12
what are, 1-2, 1-10, 3-1
prompt
logging on, 5-2
system, 5-2
protecting
accounts, 2-15
files, 5-11
programs, 4-2
PUB groups, 2-17
punctuation
commands, 5-5, 7-13
PURGE command, 7-6, 7-7, 7-8

Q

queues, 4-7, 5-13, 5-14
jobs, 8-5

R

RAM memory, 5-8
random access memory, 5-8, 5-10
read only memory, 5-9, 5-10
redirecting
information, 3-14, 3-16, 8-9
required parameters, 7-4, 7-6, 7-7, 7-9, 7-10, 7-15
RESTORE command, 5-10
restoring
information, 5-10
restricted commands, 7-2
RESUME command, 2-13
resuming
commands, 2-13
Return

keys, 6-6
what is, 5-2
ROM memory, 5-9
rules
names, 3-4, 3-6
programs, 1-9
run
implied, 4-14, 4-16
RUN command, 4-13, 4-16

S

SAVE command, 7-16
save files
capabilities, 4-7
scanner
programs, 5-5
scheduling
backing up, 5-11
jobs, 8-8
search
priorities, 4-16
security features
ACDs (access control definitiions), 4-10
segments
disks, 3-2
files, 3-4
semicolons
syntax, 7-13
sessions, 2-8
interactive, 8-4
names, 2-14, 5-6
numbers, 2-14
SF
capabilities, 4-7
shared group, 2-17
passwords, 2-17
SHOWJOB command, 8-7
SHOWME command, 3-11, 4-16, 4-17, 7-1, 7-3
SHOWTIME command, 7-3
SM
capabilities, 4-4, 7-2
software
what is, 1-2
source
files, 3-10
SPOOLF command, 5-14
spool files
files, 3-20, 5-13
input, 5-13, 5-15, 8-7
output, 5-13, 5-14
permanent, 5-14
status, 8-7
spreadsheets
programs, 1-9
SPU
hardware, 1-2
standard input
files, 3-11
standard output
files, 3-11
starting
commands, RUN, 4-13
commands, without RUN, 4-14
commands, XEQ, 4-17
jobs, 8-5
programs, 4-13, 4-14, 4-17
sessions, 2-8
statement
programs, 6-12
$STDIN
files, 3-11, 3-12
$STDLIST
files, 3-11, 3-12, 8-9
STORE command, 3-18, 5-10
storing information, 5-10
STREAM command, 8-5, 8-7, 8-8
STREAMS command, 8-5
structures
accounts, 2-3, 3-7
files, 3-9

subystems
what are, 1-6
summary
HFS syntax, 3-5
MPE syntax, 3-5
supervisor
system, 3-13, 4-4
suspending
commands, 2-13
syntax
braces, 7-12
brackets, 7-8, 7-12
commands, 7-7
commas, 7-13
diagrams, 7-7, 7-8, 7-9, 7-10, 7-12
italics, 7-12
lowercase, 7-9, 7-13
place holders, 7-13
semicolons, 7-13
uppercase, 7-11, 7-13
variables, 7-13
syntax summary
HFS, 3-5
MPE, 3-5
SYSGEN command, 3-19
SYSGEN utility, 3-14, 3-19
system
backing up, 3-19
console, 1-3
files, 3-19
manager, 3-13, 4-4
processing unit, 5-6
prompt, 5-2
recovery tapes, 3-19
supervisor, 3-13, 4-4
system-defined files, 3-11
system processing unit, 1-2

T

tape drives
devices, 1-4, 3-13, 3-18, 4-7
tapes
digital, 5-10
magnetic, 5-10
system recovery, 3-19
temporary files, 3-3, 7-8, 7-11, 7-16
terminals
console, 3-13
devices, 1-3
dialogs, 5-2
input, 3-11
output, 3-11, 7-16
terminating
commands, 2-13
text
editors, 8-1
tracks
disks, 3-3

U

UDCs, 4-11
uppercase
syntax, 7-11, 7-13
UPS
devices, 1-4
user IDs, 4-10
users
capabilities, 4-3
names, 2-3, 2-13, 2-15, 2-16, 5-6
passwords, 2-9
passwords, changing, 2-16
utilities
SYSGEN, 3-14, 3-19

V

variables
programs, 6-12
syntax, 7-13
syntax diagrams, 7-9
what are, 7-9
video screens
devices, 1-3

W

what are
  applications, 1-6
  computer language, 6-2
  computers, 1-1
  files, 3-1
  programs, 1-2, 1-10, 3-1
  subsystems, 1-6
  variables, 7-9
what is
  hardware, 1-2
  input, 1-12, 5-2
  output, 1-12
  Return, 5-2
  software, 1-2
whole numbers, 6-12
wildcards
  #, 7-4
  ?, 7-4
  @, 2-12, 7-4
  characters, 7-4
writing
  I/O, 1-12

X

XEQ command, 4-13, 4-16, 4-17