Using the BASIC 5.0/5.1 System

HP 9000 Series 200/300 Computers

HP Part Number 98613-90000

Hewlett-Packard Company
3404 East Harmony Road, Fort Collins, Colorado 80525

NOTICE

The information contained in this document is subject to change without notice.

HEWLETT-PACKARD MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MANUAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Hewlett-Packard shall not be liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

WARRANTY

A copy of the specific warranty terms applicable to your Hewlett-Packard product and replacement parts can be obtained from your local Sales and Service Office.

Copyright © Hewlett-Packard Company 1987

This document contains information which is protected by copyright. All rights are reserved. Reproduction, adaptation, or translation without prior written permission is prohibited, except as allowed under the copyright laws.

Restricted Rights Legend

Use, duplication or disclosure by the U.S. Government Department of Defense is subject to restrictions as set forth in paragraph (b)(3)(ii) of the Rights in Technical Data and Software clause in FAR 52.227-7013.

Use of this manual and flexible disc(s) or tape cartridge(s) supplied for this pack is restricted to this product only. Additional copies of the programs can be made for security and back-up purposes only. Resale of the programs in their present form or with alterations, is expressly prohibited.

Copyright © AT&T, Inc. 1980, 1984

Copyright © The Regents of the University of California 1979, 1980, 1983

This software and documentation is based in part on the Fourth Berkeley Software Distribution under license from the Regents of the University of California.

Printing History

New editions of this manual will incorporate all material updated since the previous edition. Update packages may be issued between editions and contain replacement and additional pages to be merged into the manual by the user. Each updated page will be indicated by a revision date at the bottom of the page. A vertical bar in the margin indicates the changes on each page. Note that pages which are rearranged due to changes on a previous page are not considered revised.

The manual printing date and part number indicate its current edition. The printing date changes when a new edition is printed. (Minor corrections and updates which are incorporated at reprint do not cause the date to change.) The manual part number changes when extensive technical changes are incorporated.

November 1987...Edition 1

Table of Contents

Introduction to Using the BASIC System

Chapter 1: Loading BASIC into Memory

Turn on Computer 1-2
Is BASIC Loaded and Ready for Use? 1-4
What to Do Next? 1-5
Order of Devices Searched by the Boot ROM 1-6

Chapter 2: Introduction to the System

Notations in this Manual 2-1
Using the Keyboard 2-3
What Can You Do at the Keyboard? 2-4
Performing Calculations at the Keyboard 2-4
Typing and Executing Commands 2-5
Using Typing-Aid Softkeys 2-6
Softkey Labels 2-7
What State Is the System In? 2-10
Is a Program Running? 2-10
Is There a BASIC Program in Memory? 2-13
Clearing the Program 2-14
Determining Current System Devices and Binaries 2-15
Re-Defining Typing-Aid Softkeys 2-17
Examples of Re-Defining Typing-Aid Softkeys 2-17
Softkey Labels 2-19
Memory Available for Typing-Aid Definitions 2-20
Listing the Current Typing-Aid Definitions 2-21
File for Typing-Aid Softkeys 2-21
Restoring Power-Up Typing-Aid Definitions 2-22
Defining Typing-Aid Softkeys Programmatically 2-22
Clearing the Computer 2-23
What to Do Next 2-24

Chapter 3: Mass Storage Concepts

Overview of Mass Storage Organization (or "What Are Files and Volumes?") . 3-1
A File Is Named Collection of Data 3-2
Volumes Are Collections of Files 3-3
Examples of Mass Storage Volumes 3-3

How to Specify Volumes 3-5
Checking a Disc's Format 3-9
Hierarchical Directories 3-10
What Is a Hierarchy? 3-10
Uses of the Hierarchy: An Example 3-11
Referring to Directories and Files in the Hierarchy 3-12
Choosing a Directory Format 3-13
Criteria for Choosing a Directory Format 3-14
What Is Initialization? 3-16
Disc Sectors 3-18
Sector/Volume Size Option 3-18
Disc Interleave 3-19
What to Do Next 3-20

Chapter 4: Loading and Running Programs
A Brief Look at Loading and Running Programs 4-1
A Closer Look at Loading Programs 4-4
Using LOAD 4-4
Using GET 4-5
A Closer Look at Running Programs 4-7
Prerun 4-7
Normal Program Execution 4-8
Live Keyboard 4-9
Example of Controlling Program Execution 4-9
What to Do Next 4-14

Chapter 5: Using Directories and Files
Finding and Specifying Files 5-1
Is the File on the Default Volume? 5-1
Directory, File, and Volume Specifiers 5-3
Changing the Default Volume and Current Working Directory 5-6
Creating and Using Hierarchical Directories 5-7
Example Hierarchy 5-7
Changing the Default Volume 5-7
Adding Another Directory 5-8
A Closer Look at Hierarchical Directory Capabilities 5-11
How SRM and HFS Directories and Files Are Stored 5-12
A Closer Look at File Catalogs 5-13
General File Management Operations 5-21
Closed vs. Open Files and Hierarchical Directories 5-21
Protecting Files 5-21
Copying Files and Volumes 5-32

Renaming Files 5-34
Purging Files 5-35
Volume Labels (LIF and HFS Volumes Only) 5-39
Enabling Checkread Verification 5-40
What to Do Next 5-41

Chapter 6: Editing and Storing Programs
Entering and Storing Programs (Overview) 6-1
Terminology 6-1
Getting into EDIT Mode 6-3
Correcting Typing Mistakes 6-4
Storing the Line 6-4
Entering Program Lines 6-4
Upper-case or Lower-case Letters? 6-6
Keys Used for Editing the Current Line 6-7
Keys Used for Scrolling the Program 6-9
Inserting Lines 6-10
Deleting and Recalling Lines 6-11
Copying Lines (By Changing Line Numbers) 6-12
Getting Out of EDIT Mode 6-12
Listing the Program 6-13
Storing the Program 6-13
Running the Program 6-13
A Closer Look at Editing 6-14
More Details about Getting into EDIT Mode 6-14
Typing-Aid Softkey Menu Changes (ITF Keyboards Only) 6-15
A Closer Look at Listing a Program 6-16
Global Editing Operations 6-18
Indenting a Program 6-19
Finding Textual Patterns 6-22
Search-and-Replace Operations 6-24
Copying Program Segments 6-25
Moving Program Segments 6-25
Deleting Multiple Lines 6-26
Making Programs Readable 6-27
Contrast Between Documented and Undocumented Programs 6-28
General Recommendations for Commenting Programs 6-30
Software Security 6-31
Preventing Programs from Being Listed 6-31
Other Security Measures 6-32

A Closer Look at Storing Programs 6-33
Using STORE 6-34
Using SAVE 6-34
What to Do Next 6-36

Chapter 7: ITF Keyboards
BASIC ITF Keyboard Overlays 7-2
Character Entry Keys 7-2
Cursor-Control Keys 7-5
Numeric Keypad 7-6
Editing Keys 7-7
Program Control Keys 7-9
System Control Keys 7-10
Softkeys and Softkey Control 7-12

Chapter 8: HP 98203B/C Keyboards
Character Entry Keys 8-2
Numeric Keypad 8-4
Cursor-Control Keys 8-5
Editing Keys 8-6
System Control Keys 8-9
Softkeys 8-11
Program Control Keys 8-12

Chapter 9: HP 98203A Keyboards
Character Entry Keys 9-2
Cursor-Control Keys 9-4
Editing Keys 9-5
System Control Keys 9-8
Softkeys 9-10

Index

Introduction to Using the BASIC System

This manual contains information about using the BASIC system that you may use frequently. If you need information about installing BASIC onto a hard disc, configuring or customizing your system, maintaining BASIC (making back-ups, etc.), you should reference the *Installing and Maintaining the BASIC System.*

Here is what you will find in this manual:

Chapter	Description
"Loading BASIC into Memory"	a chapter that describes loading an installed BASIC system
"Introduction to the System"	a chapter that shows you the keyboard operations used in BASIC, and how to read the display during a BASIC session
"Mass Storage Concepts"	a chapter that describes detailed concepts related to mass storage (storing files, etc., on discs)
"Loading and Running Programs"	shows you how to load programs from a disc, and run them
"Using Files and Directories"	a chapter that explains how you specify files in BASIC, and how to use an HFS (Hierarchical File System) directory structure
"Editing and Storing Programs"	when you need to write programs, go to this chapter for information on the BASIC editor
"Keyboard" chapters	for detailed information about common keyboards and their use with BASIC.

Tasks/Topics	Where to Go For Details
If you have **not** installed BASIC...	*Installing and Maintaining the BASIC System*
Load BASIC from back-up discs or from hard disc...	1-1
[illegible]	[illegible]
If loading fails...	1-5
What to do next...	1-5
Order of devices searched by the Boot ROM...	1-6

Loading BASIC into Memory 1

This chapter **is not for installing BASIC**. You should have already installed BASIC in *Installing and Maintaining the BASIC System.* If you have not done so, go to that manual now.

If your System Administrator (the person who installed your system and possibly configured it) has an alternate loading procedure, use that procedure instead of the following *default*[1] loading procedure.

A detailed description of how to choose a system to load is included in this chapter. If you need more information, see the appropriate section in *Installing and Maintaining the BASIC System* (the one you used when installing).

Table 1-1. Loading BASIC

If You Have This Configuration...	Do These Steps...	See Page...
Only flexible disc drives	1. Make sure computer and disc drives are turned off 2. Turn on one flexible disc drive 3. Insert back-up copy of System DISC ONE 4. Turn on computer (be sure monitor is on)	1-5
One or more hard disc drives (and you do **not** have an HP-UX system currently running)	1. Make sure computer is turned off 2. Hold down spacebar and turn computer on 3. Choose a system (for example, `SYSTEM_BA5`) by typing characters to left of the system entry on the screen; e.g., `1B`	1-2
One or more hard disc drives and an **HP-UX system is currently running**	1. **Do not turn off the computer or disc**: See the System Administrator who will shutdown the system. 2. Only after the system has been shutdown, reboot the system (`reboot`) and hold down the spacebar 3. Choose a system (such as `SYSTEM_BA5`) by typing characters to left of the system entry (e.g., `1B`)	1-2

[1] A default procedure is one that is standard or assumed by the system if not changed by a user.

Turn on Computer (and Hold Down Spacebar if You Have Other Systems On-Line)

If there are **no** other on-line systems, you **need not** hold down the space bar (the boot ROM searches for your system and loads the default system automatically).

After pressing the computer's power switch on (or rebooting), press the space bar on the keyboard and hold it down.

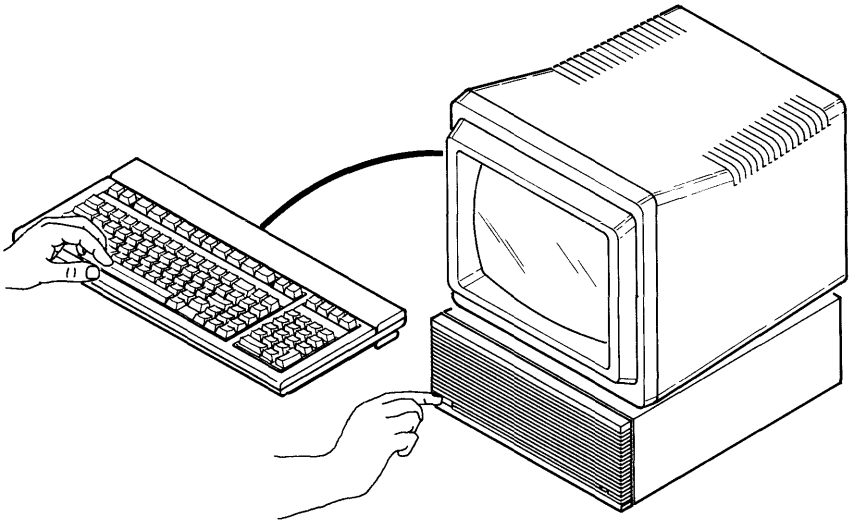


Figure 1-1. Turn Computer On and Hold Down the Space Bar

Holding down the space bar does two things:

- Tells the computer to wait until you are ready to load a system (instead of automatically loading the first one it finds).
- Allows you to watch the computer's display as it:
  - performs its self-test (checks memory, etc.)
  - searches for interfaces (such as HP-IB, RS-232C, etc.), and *on-line systems* (operating systems stored on discs, tapes, and ROM cards). For a list of the order in which the devices are search, see the last section in this chapter, "Order of Devices Searched by the Boot ROM."

If You See Nothing on the Monitor Screen

Here are some possible explanations if there is nothing on the monitor screen:

- The monitor's brightness is not turned all the way up.
- The monitor is not plugged in.
- The computer is not plugged in.

If You Still Have Problems

If you have verified that the above problems do not exist, but you are still not getting a display, see your computer's *Installation Reference* for troubleshooting advice.

Choosing a System

Type the number and letter to the left of the "SYSTEM_BA5" entry. For example in the figure below, you would type [1] [B]; the lower right shows the characters you type.

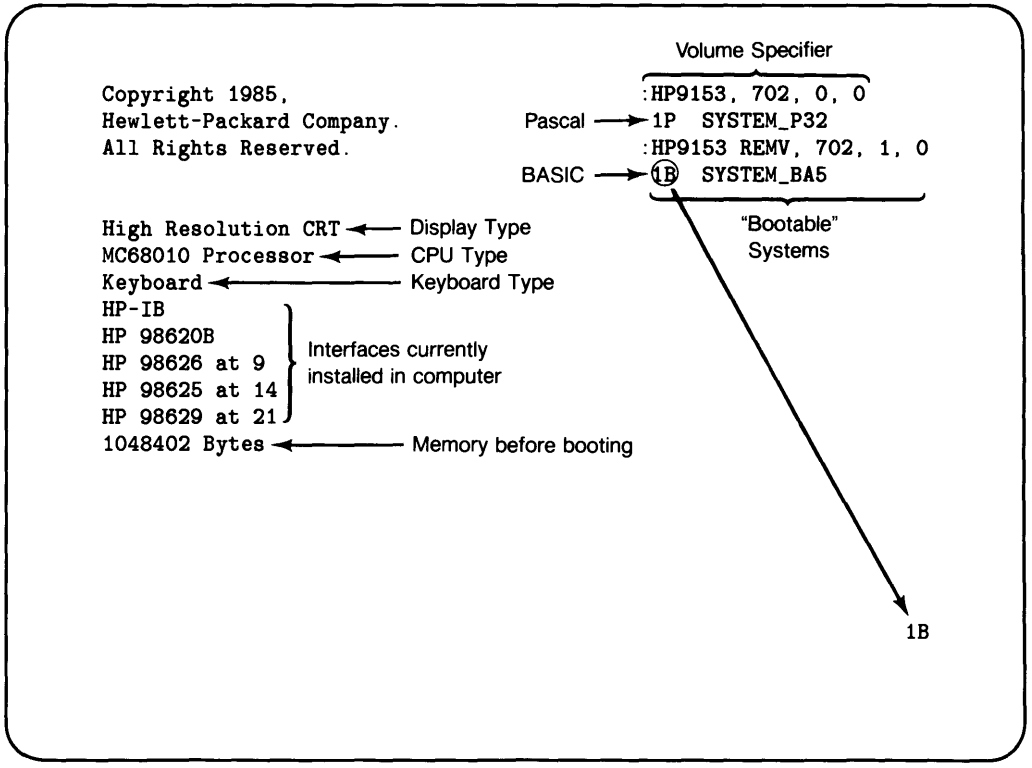


Figure 1-2. Choosing a System

The lower-left corner of the display will show the following message to indicate that it is loading the specified system:

```
BOOTING A SYSTEM
RESET To Power-up
```

See the end of this chapter for "Order of Devices Searched by the Boot ROM". If you loaded BASIC from a hard disc, you have finished. See "What to Do Next."

Is BASIC Loaded and Ready for Use?

If you are loading BASIC from a back-up copy of the System flexible discs, you will switch discs until the AUTOST program displays this message:

```
The BASIC system is now ready for use.
```

BASIC has been successfully loaded. You are ready to use BASIC; see "What to do Next" for more information.

If you are booting a STOREd SYSTEM from a hard disc or SRM, you will see the message:

```
BASIC Main 5.1
```

To verify you have all the binaries loaded, run:

```
LIST BIN [Return]
```

If you loaded from a customized AUTOST program, see your System Administrator for details.

If Loading Fails

If the computer did **not** load the BASIC system, then you should check for the following potential problems.

Problem	Solution
The Boot ROM does not find the BASIC system	Make sure that the disc drive is plugged in and turned on. Make sure that the drive is connected to the HP-IB interface of the computer. Make sure the back-up System disc is properly inserted in the drive.
`NOT ENOUGH MEMORY` is displayed	Your computer does not have sufficient memory, or memory board(s) have been improperly installed. Refer to the *Peripheral Installation Guide* for instructions on how to configure and install memory cards.

What to Do Next?

Now that BASIC is loaded, you may begin using BASIC. The following table shows some chapters for more information on BASIC.

Topic	Chapter
Learn about some BASIC fundamentals	"Introduction to the System"
Learn about mass storage devices	"Mass Storage Concepts"
How to run BASIC programs	"Loading and Running Programs"
How to manage files	"Using Files and Directories"
How to edit programs	"Editing and Storing Programs"

Order of Devices Searched by the Boot ROM

The following table lists the order in which the Boot ROM searches mass storage devices for system files.

Priority	Type of Mass Storage Device	Comments
1	Internal disc drive (9826 and 9836 only)	Select code 4; unit number 0 (right drive)
2	External disc drives	Select codes 7 through 31; primary address 0; unit number 0; volume number 0
3	Shared Resource Manager (SRM)	Select code 21; node number 0; volume number 8; `/SYSTEMS` directory
4	Bubble memory card (HP 98259)	Select code 30 (Bubble memory is organized as files and treated the same as other mass storage devices.)
5	EPROM card (HP 98255)	Unit 0 (EPROM is also organized and treated like other mass storage devices: if a system is found in EPROM, then it is first transferred to computer memory and executed from there—the CPU does not execute the system directly from EPROM)
6	ROM-based operating systems	Execute directly from ROM and do not have to be loaded into computer memory
7	Internal disc drive (HP 9826 and HP 9836 only)	Select code 4; unit number 1 (left drive)
8	Remaining external disc drives	Select codes 7 through 31; addresses 0 through 7; unit numbers 0 through 7; volume numbers 0 through 7
9	Remaining SRM systems	Select codes 8 through 31
10	Remaining bubble memory cards	Select codes 0 through 31, except 30
11	Remaining EPROM units	Units 1 through last one found

Tasks/Topics	Where to Go For Details
Notations in this Manual	2-1
Using the Keyboard	2-3
What Can You Do at the Keyboard?	2-4
Performing Calculations at the Keyboard	2-4
Typing and Executing Commands	2-5
Using Typing-Aid Softkeys	2-6
Softkey Labels	2-7
What State is the System in?	2-10
Is a Program Running?	2-10
Is There a BASIC Program in Memory?	2-13
Clearing the Program	2-14
Determining Current System Devices and Binaries	2-14
Re-Defining Typing-Aid Softkeys	2-17
Softkey Labels	2-19
Memory Available for Typing-Aid Definitions	2-20
Listing the Current Typing-Aid Definitions	2-21
File for Typing-Aid Softkeys	2-21
Restoring Power-Up Typing-Aid Definitions	2-22
Defining Typing-Aid Softkeys Programmatically	2-22
Clearing the Computer	2-23
What to Do Next	2-24

Introduction to the System 2

This chapter introduces you to using the keyboard and display of HP Series 200/300 computers. Topics include how to type in and execute commands, use typing-aid softkeys, determine how much memory is available, and determine which device is the system printer.

Notations in this Manual

The following table describes some of the notations (conventions) used in this manual.

If You See...	It Means...
`COMPUTER FONT`	this is either what you see as the system's response to your commands, or this is exactly what you should type in an example
italic font	when you see examples with italics in them, you have to replace the italic words with your own (i.e., if the italic word is *file_name*, then you supply a real file name in place of *file_name*)
[Break]	if you see a word in a box, it refers to an actual key on your keyboard; for example, look on your keyboard for [Break] (upper left on the ITF keyboard)
[Shift]-[Break]	when a key is prefaced with [Shift]-, it means you press the [Shift] key, hold it down and press the next key (like shifting case)
`Caps Lock On`	if you see this in the System Message/Results line, it means you will type upper-case letters
`Caps Lock Off`	if you see this in the System Message/Results line, it means you will type lower-case letters
`Caps`	(ITF keyboard only, at lower right of screen) means you will type upper-case letters
(*blank*)	(ITF keyboards only, at lower right of screen) means you will type lower-case letters

Throughout this manual, the key cap [Return] will be used. If you have a keyboard other than the ITF keyboard (with a key like [Enter] instead of [Return]), substitute the appropriate key.

The Significance of Letter-case

While typing commands, letter-case is *usually* not important. For instance, these two commands are recognized as being equivalent by the system as the BEEP statement:

```
beep
BEEP
```

However, letter-case *is* significant when typing in things like literal text (enclosed in "quotes"). Also, BASIC keywords can be all upper-case or all lower-case; however, when letter-case is **mixed** in a keyword, the system interprets it as a variable name. For instance, `Beep` would be interpreted as a variable name, not as a keyword. Here are some examples that are **not** equivalent:

Table 2-1. Example Commands that Are NOT Equivalent

First Command	Second Command	Reason not Equivalent
`LOAD "MyFile"`	`LOAD "MYFILE"`	Literal string
`system$("MSI")`	`system$("msi")`	Literal string
`BEEP`	`Beep`	Can't mix letter-case in keywords

Using the Keyboard

There are three different types of keyboards available with Series 200 and Series 300 computers. Here are the options, along with the names by which we will refer to them throughout the BASIC manual set.

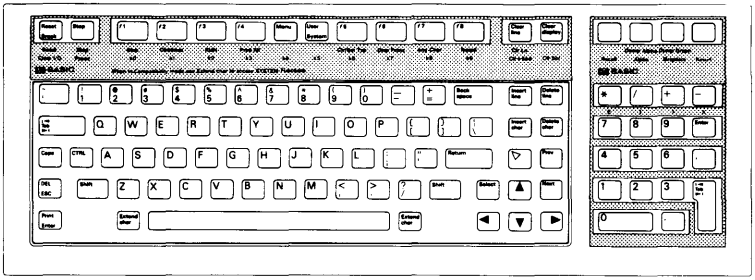


Figure 2-1. ITF Keyboard (with BASIC Keyboard Overlay)

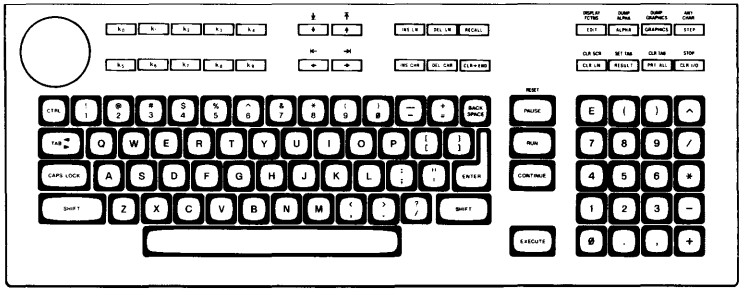


Figure 2-2. HP 98203B/C Keyboards

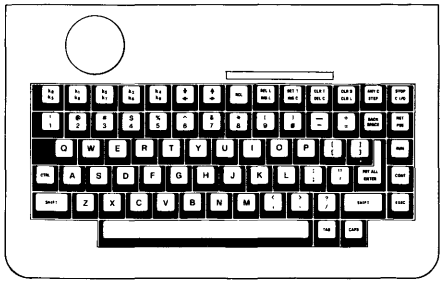


Figure 2-3. HP 98203A Keyboard

What Can You Do at the Keyboard?

There are several things you can do at the keyboard of this system:

- perform calculations
- type in and execute commands
- load and run programs, and control program execution
- type in, edit, and store programs.

Let's now look at how to perform calculations, and type in and execute commands. (The last two of these operations are discussed in subsequent chapters.)

Performing Calculations at the Keyboard

BASIC has a command interpreter that can also evaluate numeric expressions. For example, you could type:

```
99/9 [Return] ←—— Characters you type appear here.

11 ←————————— System response appears here.
```

Note that with HP 98203 keyboards, you will press the [ENTER] key instead of the [Return] key.

You can use arithmetic operators such as

+ for addition
- for subtraction
/ for division
* for multiplication

as well as parenthesis, and exponentiation (^). For a list of priority when evaluating an arithmetic expression, see *Programming Techniques, Volume 1: General Topics*, "Numeric Computation," for a table listing the hierarchies.

Typing and Executing Commands

You can type in and execute commands from the keyboard at all times except when:

- there is currently a command being executed, with another one already entered and waiting to be executed
- there is a program running that traps keystrokes (with ON KBD) or disables the keyboard (with SUSPEND INTERACTIVE).

At all other times, you can type in commands and press Return to present them to the system for execution. The system parses the command and takes the appropriate action.

Example Command (Determining Available Memory)

Type in and execute the following command (characters will appear in the "Keyboard Input" line near the bottom of the display):

```
SYSTEM$("AVAILABLE MEMORY")  Return
```

The system returns something like this in the "System Messages" line at the bottom of the display:

```
123456
```

This value represents the number of unused bytes of memory.

Example Command (Checking and Setting the System Clock)

The following functions allow you to check the setting of the system clock (the CLOCK binary must be already loaded):

```
DATE$(TIMEDATE), TIME$(TIMEDATE)  Return
```

The system returns something like this:

```
 17 Mar 1987       10:27:32
```

You can set the time and date with the SET TIMEDATE statement; here is an example:

```
SET TIMEDATE DATE("17 Mar 1987")+TIME("10:30:00")  Return
```

If you are sharing a hard disc with an HP-UX system, you should also use the TIMEZONE IS statement for compatibility with time stamps on files. See the *BASIC Language Reference*, TIMEZONE statement, for details.

Using Typing-Aid Softkeys (Requires KBD Binary)

There are "softkeys" on all three of the keyboards available on Series 200/300 computers. Here are the locations of these keys:

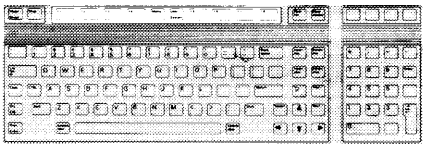


Figure 2-4. Location of Softkeys on ITF Keyboard

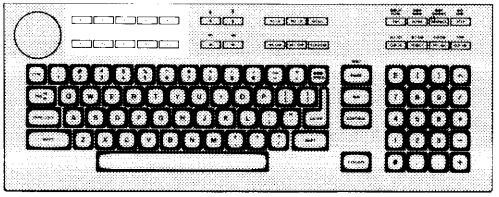


Figure 2-5. Location of Softkeys on 98203B/C Keyboard

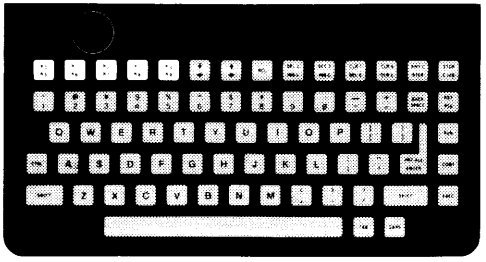


Figure 2-6. Location of Softkeys on 98203A Keyboard

Softkey Labels

Before using any of the softkeys, be sure the one you choose to press is the one you really want. (On an ITF keyboard, you can press [Menu] to turn on the softkey labels.) To see the current definitions of the softkeys, execute the following command (KEY LABELS requires CRTX binary):

```
KEY LABELS ON [Return]
  or
CONTROL CRT,12;2 [Return]
```

This command turns on the "Key Labels" area at the bottom of the display as shown below (this example is for a system with an ITF keyboard):

```
                                Print      Clr Tab  Display  Any
Step     Continue RUN           All        Set Tab  Fctns    Char     Recall
```

The following diagram shows the format of the softkey labels when an HP 98203 keyboard is in use:

```
                RENumber      FIND ""       CHANGE "" TO " INDENT
SCRATCH         LOAD ""       LOAD BIN ""   LIST BIN       RE-STORE ""
```

Example Typing-Aid Softkey Definition

Typing-aid softkeys produce a sequence of characters[1] on the keyboard input line (or on the "current line" in EDIT mode). For instance, pressing [f1] (on an ITF keyboard) produces these characters:

```
EDIT
```

You can also re-define these keys, so each produces a special set of characters specific to your needs. See the "Re-Defining the Typing-Aid Softkeys" for details.

[1] Typing-aid softkeys are only active when a running program has not defined them to produce an interrupt. See "Program Structure and Flow", *Programming Techniques* for ON KEY details.

Softkey-Menu Indicator (ITF Keyboard Only)

On computers with ITF keyboards, the softkeys ([f1] through [f8]) have **four independent sets of definitions.** You can select which set of definitions (and labels) to activate by using the [System], [User] ([Shift]-[System]), and [Shift]-[Menu] keys. You can turn the labels on and off with the [Menu] key.

With each set of definitions, there is a menu with current definitions.

The System softkey menu (press [System]) shows definitions for "immediate-execute" system operations such as Step, RUN, and Recall. When this menu is active, the keys [f1] through [f8] have the definitions shown below:

```
                                                        System  Caps     Idle

                                Print        Clr Tab  Display  Any
Step      Continue RUN          All          Set Tab  Fctns    Char     Recall
```

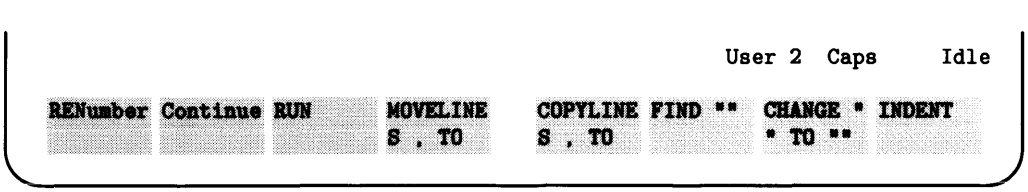
The System menu is the default menu at system power-up, after a SCRATCH A, etc. (when the KBD binary is not present).

The User 1 softkey menu (press [User]) shows typing-aid definitions for the User 1 softkeys. These keys will produce specific sequences of characters when pressed. For instance, pressing [f1] will produce the characters `EDIT` while in this menu.

```
                                                        User 1  Caps     Idle

EDIT      Continue RUN          SCRATCH      LOAD ""  LOAD BIN LIST BIN RE-STORE
                                                      ""                 ""
```

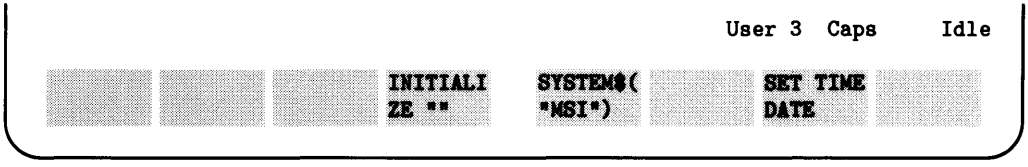
The User 1 menu is the default menu when the KBD binary is loaded. If you are in the System menu and press [User], you will be returned to the User menu (but not necessarily User 1). If you are in one of the User menus and press [User], you return to the User 1 menu.

The User 2 softkey menu (press [Shift]-[Menu]) has the following default definitions:



Softkeys [f4] through [f8] are only defined if the PDEV and EDIT binaries are loaded at the time that the default definitions of these keys are set up (when SCRATCH A, LOAD KEY, or LOAD BIN "KBD" are executed).

The User 3 softkey menu (press [Shift]-[Menu] from User 2 menu) has the following default definitions:



Pressing [Shift]-[Menu] cycles you through the User menus (from 1 to 2 to 3 to 1, and so on).

What State Is the System In?

When BASIC is booted, memory is cleared and various system elements are assigned default values. For example, the CRT display is assigned as the system printer. This condition is called the "power-on state"[1].

If your computer has been used since power-on, it may be in a somewhat unknown state. For instance, there may be an unwanted program in memory, or the default printer or volume may have been changed to specify devices that you don't want. This section explains:

- how to find out what your computer is doing, and what state it is in
- how to get it into a state in which you can begin to use it properly.

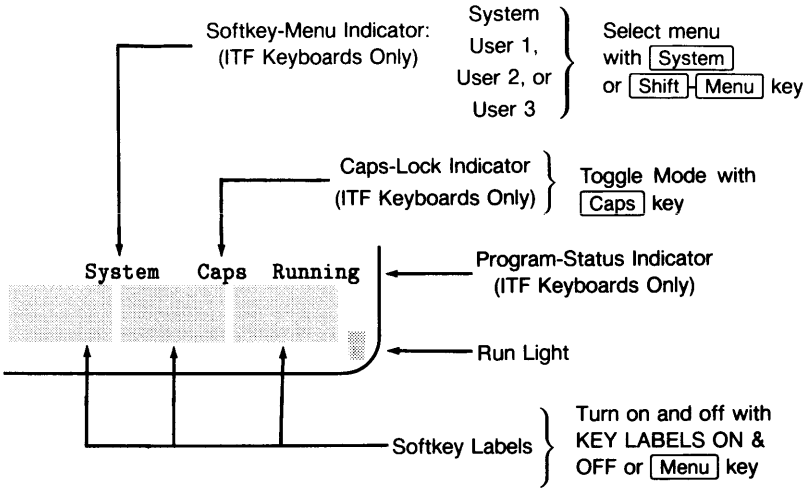
Is a Program Running?

If there are several people sharing a computer, you should generally find out whether the computer is currently in use so you will not destroy someone else's work. It is also convenient to be able to quickly glance at the display to find out whether your own program is running, waiting for input, or for a device to become available, and so forth.

Program-Status Indicators

You can determine the current status of the system by looking at the lower right-hand corner of the CRT. BASIC uses this area to display information about whether a program is currently running, what softkey menu is currently active, and so forth.

[1] For a complete list of power-on defaults, see the "Useful Tables" appendix of the *BASIC Language Reference*.



The character in the lower right corner is referred to as the "run light". The Table 2-2 shows the various indications of the run light and their meaning.

Table 2-2. Run Light Indications

Status Indicator[1]	Run Light	System State
`Idle`	*(blank)*	Program stopped; can execute commands; CONTINUE not allowed.
`Running`	▒	Program running; can execute commands. CONTINUE not allowed.
`Paused`	_	Program paused; can execute commands; CONTINUE *is* allowed.
`Transfer`	IO	Program paused, but an overlapped TRANSFER (I/O) operation is still in progress; can also execute commands.
`Input?`	?	BASIC program waiting for input from keyboard; cannot execute commands.
`Command`	*	System executing command entered from keyboard; can enter 1 more command, but it will not be executed until after the current command is completed.

[1] These indicators are displayed only if softkey labels are currently on. Use the KEY LABELS ON statement (or the Menu key on an ITF keyboard) to turn these labels on.

Pausing, Stopping and Continuing Programs

If the computer operator does not intervene, a program will run until it reaches an END, STOP, or PAUSE statement, or until it pauses to input data or report an error. If you wish to pause or stop a program before its normal completion, continue operation, or abort an I/O statement, use the following keys:

Table 2-3. Pausing and Stopping Programs

98203B/C Keyboard	ITF Keyboard	98203A Keyboard	Effect
[PAUSE]	[Stop] (Pause)	[PSE]	Pauses program execution after the computer finishes the line it is on and any I/O operations in progress. This is useful for pausing a program that is executing an INPUT statement. It leaves all necessary internal information intact, so that program execution can be resumed again with the index INPUT statement the [CONTINUE] key (see below) or CONT command.
[CONTINUE]	[f2] (Continue)	[CONT]	Pressing [CONTINUE] after a PAUSE (key or statement) causes program execution to resume in a normal manner from the place where it was paused.
[CLR I/O]	[Break]	[C I/O]	Aborts any I/O statement in progress (such as ENTER or TRANSFER) and pauses the program. The program counter is returned to the beginning of the aborted I/O statement, so that the [CONTINUE] key or the CONT command cause the program to resume program execution beginning with that same statement. This is useful when the computer is "hung" trying to output to a device that is down.

Table 2-3. (Continued)

98203B/C Keyboard	ITF Keyboard	98203A Keyboard	Effect
[STOP] ([SHIFT]-[CLR I/O])	[Stop] ([Shift]-[Stop])	[STOP] ([SHIFT]-[C I/O])	Stops the program after the computer finishes executing the line it is on (and returns the program to the main context, if executing a subprogram or user-defined function). However, it does not affect the interfaces, the CRT, program memory, the values of variables, tabs, or the [RECALL] key's buffer. [CONTINUE] is not allowed after a [STOP].
[RESET] ([SHIFT]-[PAUSE])	[Reset] ([Shift]-[Break])	[RST] ([SHIFT]-[PSE])	This is the most drastic and complete way to halt program execution. The program stops immediately, all I/O operations are aborted, any open files are closed, and all interface cards are reset. However, the printout area of the CRT, program or variable memory, tabs, and the [RECALL] key's buffer are not affected. [CONTINUE] is not allowed after a [RESET].

The easiest way to remember the key definitions on the ITF keyboard is to use the keyboard overlay and keep the softkey labels on. (To turn softkey labels on, either press the [Menu] key or execute the KEY LABELS ON statement.)

Is There a BASIC Program in Memory?

If a BASIC program is currently in memory, then use this command to list the lines on the current *system printer* (the default is the display screen[1]):

```
LIST [Return]
```

If there is a program in memory, you will see something like this:

```
10  PRINT "Short program."
20  PRINT "Good-bye."
30  END
```

[1] The subsequent section called "Determining Current System Defaults" shows how to determine which device is the current system printer.

After the listing is complete[1], the system displays the amount of memory currently available for BASIC programs and variables at the lower left corner of the screen:

```
Available memory = 5629926
```

(If there is no program currently in the computer, then the amount of available memory is all that will be displayed by LIST.)

Clearing the Program

Before you clear a program, you might want to determine what the current defaults are so you can reset them if they are modified. Clear the computer's memory with the SCRATCH command (see "Clearing the Computer"). The next section provides instructions for determining these defaults.

[1] If listing the program takes longer than you want to wait, you can stop it by pressing the Break key (CLR I/O on 98203B/C keyboards; CI/O on 98203A keyboards).

Determining Current System Devices and Binaries

You can determine the current state of several of these system defaults by using the following statements:

Table 2-4. System Defaults

Method	Explanation	Default
`SYSTEM$("PRINTER IS")`	returns the select code of the current "system printer" (the destination of PRINT operations).	`PRINTER IS CRT`
`SYSTEM$("PRINTALL IS")`	returns the select code of the current "printall printer" (the destination of system messages when PRINTALL ON is active).	`PRINTALL IS CRT`
`SYSTEM$("DUMP DEVICE IS")`	returns the select code of the current "system dump device" (the destination of DUMP ALPHA and DUMP GRAPHICS operations).	`DUMP DEVICE IS 701`
`SYSTEM$("MSI")`	displays the current "default drive" (the mass storage device that will be used when one is not explicitly specified).	device from which you booted the BASIC system[1]
`SYSTEM$("AVAILABLE MEMORY")`	returns the amount of memory available for application programs and their data.	*(not applicable)*

[1] If you have ROM-based BASIC, this device will be the first mass storage device found with storage media present. See the "Order of Devices Searched by the Boot ROM" section of the "Loading BASIC into Memory" chapter for a list of the devices searched by the Boot ROM.

Table 2-4. (Continued)

Method	Explanation	Default
`SYSTEM$("VERSION:BASIC")`	returns the revision number of the BASIC system.	*(not applicable)*
`SYSTEM$("VERSION:ERR")`	returns the revision number of the ERR binary (if present), or returns 0 (if the binary is not present). To check for other binaries, substitute the binary's name for `ERR` in the `SYSTEM$` function call; for instance, `SYSTEM$("VERSION:EDIT")` would return the version number of the EDIT binary, if currently loaded, or 0 if it is not loaded.	*(not applicable)*
`LIST BIN`	displays a list of **all** language extension and driver binaries currently residing in memory. (For further information about language extension and driver binaries, see the "Language Extensions, Drivers, and Configuration" chapter in *Installing and Maintaining the BASIC System.*)	`CRTA` and `CRTB`

See the the *BASIC Language Reference* for further capabilities of SYSTEM$; see *Installing and Maintaining the BASIC System* for information on LIST BIN.

Re-Defining Typing-Aid Softkeys

The default typing-aid softkey definitions are useful, but you may want to define a key that is more specific to your own needs. This section describes how to change the current definitions of typing-aid keys; it also shows how to store these definitions in a file so you can programmatically load them at a later time.

KBD Binary Is Required

In order to re-define typing-aid keys, make sure the KBD binary is currently loaded (see the preceding section for examples of LIST BIN and SYSTEM$ statements to load and use LOAD BIN).

Examples of Re-Defining Typing-Aid Softkeys

The first example shows how to re-define softkey [f1] ([k1]) to produce the characters "My very own keystrokes" on the display whenever you subsequently press this key. Remember that these examples use the ITF keyboard; if you have another keyboard, there will be slight differences (such as the availability of editing [k0] instead of [f1]).

Example 1

1. Enter the edit-softkey mode for the desired softkey. There are two ways to enter this mode; this example describes one way, and the next example describes the other.

   On an ITF keyboard, get into the `User 1` softkey menu and press the EDIT key ([f1]), and then press [f1] again followed by [Return]. (On a 98203 keyboard, press the [EDIT] key, and then press [k1] followed by [ENTER].)

   The system then displays the key's current definition (on the keyboard input line), followed by a message to indicate you can now modify the definition of the softkey:

```
K#EDIT <-------- Displayed on the keyboard input line.

Editing key 1 <-- Displayed on the system message line.
```

(The keyboard input line is blank if the key currently has no definition.)

2. Press [Shift]-[Clear line] to clear the key's current definition.
3. Enter the new definition.

   Type the desired characters on the keyboard input line. In this example, type:

   ```
   My very own keystrokes.
   ```

4. Exit the softkey-edit mode.

   After you have finished modifying the softkey's definition, press [Return] to exit the softkey editing mode and update the softkey's definition. If you don't want to update this softkey's definition, press [PAUSE] ([Stop] on an ITF keyboard) to abort the re-definition and retain the existing definition.

5. Verify that the key works as desired.

   Press the softkey [f1] ([k1]). The following characters: `My very own keystrokes.` should be reproduced exactly as you typed them. (If not, go to step 1 and try again.)

6. Press [Shift]-[Clear line] to clear the line for the next example.

Example 2

Let's now suppose that you want to define a typing-aid key that: clears the line you are on, types a command, and then executes that command (this is how the RUN and CONTINUE softkeys are defined by the system). Here is what you could type:

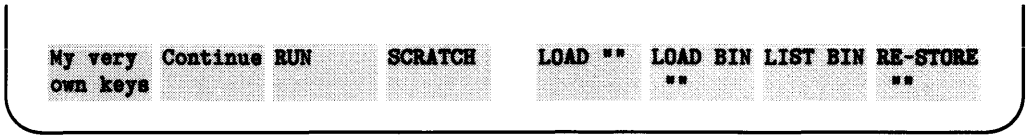
```
EDIT KEY 2 [Return]

[Shift]-[Clear line] [CTRL]-[Shift]-[Clear line] LIST [CTRL]-[Return] [Return]
```

The notation [CTRL]-[Return] indicates that you are to hold down the [CTRL] key and then press the [Return] key, then release the [Return] key followed by releasing the [CTRL] key. The [CTRL] key tells the system that you don't want to execute that key's function, but that you want it to produce a sequence of characters on the screen. The sequence begins with the inverse-video "K" character, CHR$(255), followed by another character; for instance, [CTRL]-[Shift]-[Clear line] produces "K#", and [CTRL]-[Return] produces "KE".

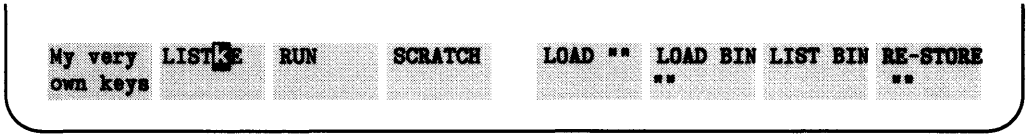
After you enter your softkey definition for key 2, pressing key 2 will execute the LIST command. **If you include a key preceded by [CTRL] (as in the above [CTRL]-[Return]) the following key will be executed when you press the softkey.** For example, if you tack a [CTRL]-[Return] at the end of your softkey definition, [Return] will be executed as part of the softkey function.

Softkey Labels

You may have noticed the two different softkey labels produced in the preceding examples. The label produced in the first example should have begun with the characters `My very own keys` (the rest of the characters in the softkey's definition are not shown in the label).



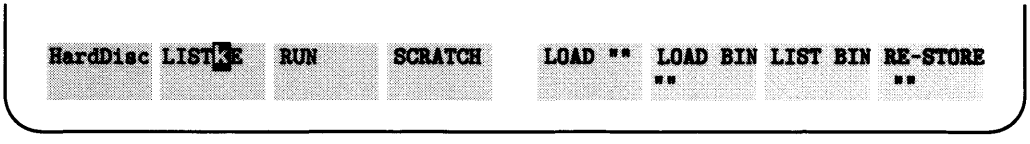
However, the label of the second example was simply `LIST`.



You can also use a similar effect of the [Clear line] key in softkey labels to have nice-looking labels that are not just the first *n* characters in the key's definition (in which *n* is the width of the softkey label). For instance, suppose that you want to have a key that produces the characters `MSI ":CS80,700,0,1"`, but you would rather the label be `HardDisc`. Here is what you would enter in the softkey's definition:

```
EDIT KEY 1 [Return]
HardDisc        [Shift]-[CTRL]-[Clear line] MSI ":CS80,700,0,1" [CTRL]-[Return] [Return]
```

The label `HardDisc` would be displayed momentarily whenever the key is pressed, but the [Clear line] keystroke would soon erase the characters, executing the `MSI ":CS80,700,0,1"` command. (The spaces following `HardDisc` move the "k#" characters out of the label.)



Example 3

When you use some of the softkey menus, you will see how some softkeys not only echo a command onto the command line, they enable you to begin inserting within quotes. Example 3 shows you how to do this with the MOVELINES command.

1. `EDIT KEY 1` [Return]
2. [Shift]-[Clear line] (*to clear the line*)
3. [CTRL]-[Shift]-[Clear line] `MOVELINES "","" TO ""` [k H][k>][k>][k>][k>][k>][k>][k>][k>][k>][k>][k>][k>][k+]

   where [k H] represents: [CTRL]-[Shift]-[◄]
   [k>] represents: [CTRL]-[►], and
   [k+] represents: [CTRL]-[Insert char]
4. Press [Return]

Now, when you press key 1, the command line prints the MOVELINES command and lets you insert in the first double quotes. The [k H] moves the cursor to the beginning of the line; the [k>] moves the cursor 11 spaces to the right (to the first set of double quotes), and the [k+] puts you in the insert mode.

Memory Available for Typing-Aid Definitions

There is approximately 1024 bytes of memory used by the system for the purpose of storing the typing-aid softkey definitions. Since there is a small amount of overhead required for each key, there is actually only about 1000 bytes available for keystrokes.

The maximum number of characters you can put into each definition is 256 characters. However, if you produce a 256-character typing-aid softkey definition—on a system with a high-resolution display and then try to *edit* it on a system with a medium-resolution display, you will only be able to get 160 characters into the modified typing-aid definition. There is also an additional restriction: attempting to *use* a softkey definition that contains more characters than will fit into the Keyboard Input Line will result in lost characters.

Listing the Current Typing-Aid Definitions

You can list all current typing-aid softkey definitions by executing this statement:

```
LIST KEY
```

The keys are listed on the current default printer (usually the CRT).

You can also list them on another device by specifying that device's select code. For instance, to list them on a printer, execute a command something like one of the following:

```
LIST KEY #PRT
  or
LIST KEY #701
```

Since most printers cannot print the inverse-video "k" that would show up in the keys' definitions, LIST KEY substitutes the letters **System key**: for this character when the keys are displayed. For instance:

```
Key 2:
System key: #          Clear line key.
LIST
System key: E          Return key.
```

File for Typing-Aid Softkeys

If you have modified any of the softkeys, you might want to store the new definitions somewhere. With the STORE KEY statement, you can store all of the current typing-aid softkey definitions in a file. Use LOAD KEY to subsequently load these definitions back into the computer.

Storing the Current Typing-Aid Softkey Definitions in a File

If you want your current typing-aid softkey definitions to be stored in a file called "MyKeys", you could use this command:

```
STORE KEY "MyKeys"
```

If the file already exists, you must use this command:

```
RE-STORE KEY "MyKeys"
```

Loading Typing-Aid Softkey Definitions from a File

To load the typing-aid softkey definitions stored in the preceding example, execute this command:

```
LOAD KEY "MyKeys"
```

Restoring Power-Up Typing-Aid Definitions

To restore the power-up default definitions, execute the statement without specifying a file:

```
LOAD KEY
```

Defining Typing-Aid Softkeys Programmatically

You can also write a program that defines the typing-aid softkeys using the SET KEY statement. See the "Communicating with the Operator" chapter of *BASIC Programming Techniques* for details.

Clearing the Computer

As discussed in the preceding section, your computer may or may not be in a desirable state.

- If it is in an undesirable state, then you can "clear" it as described in this section.
- If everything is acceptable, then you can skip this section and go to "Loading and Running Programs".

When power is first switched on and the BASIC system is booted, computer memory is cleared and various system elements are assigned default values. Turning power off and then back on again is one way to clear the computer. However, this is not necessary and often is not convenient. An easier and faster way is to use SCRATCH commands.

There are several forms of this command to allow a choice of clearing actions. The following paragraphs give general descriptions of the choices; complete descriptions are in the *BASIC Language Reference* under SCRATCH in the keyword dictionary and in the "Reset Tables" of the "Useful Tables" appendix.

`SCRATCH R` clears the [RECALL] key's buffer.

`SCRATCH KEY` clears typing-aid softkey definition(s). See the descriptions of typing-aid keys in preceding sections of this chapter for further information.

`SCRATCH C` clears **all** variables from the computer's memory, including COM variables. However, the current program and softkey definitions are left intact.

`SCRATCH` clears all program lines currently in the computer's memory. It also clears all variables which are not in COM. See the "Subprograms" chapter of *BASIC Programming Techniques* for a description of COM.

`SCRATCH A` clears most everything from the computer's memory, restoring the system to its power-on state. The only exceptions are the [RECALL] key's buffer, the real-time clock, and the currently loaded binaries.

`SCRATCH BIN` removes all of the current binaries in memory (except the driver of the currently active CRT display), and re-executes all of the steps in a power-up operation (except loading and running the "autostart" program). See the "Language Extensions, Drivers, and Configuration" chapter of *Installing and Maintaining the BASIC System* for details.

What to Do Next

Task/Topic	Chapter
Learn about the BASIC mass storage system	"Mass Storage Concepts"
Learn how to use and manage files and directories	"Using Files and Directories"
Learn how to load and run programs	"Loading and Running Programs"
Learn how to enter, edit, secure, document, and store programs	"Editing and Storing Programs"

Steps in Installation	Where to Go For Details
Overview of mass storage organization ("What are files and volumes?")	3-1
How to specify volumes	3-5
Checking a disc's format	3-9
Hierarchical directories	3-10
Choosing a directory format (LIF vs. HFS)	3-13
What is initialization?	3-16
Disc sectors	3-18
Disc interleave	3-19
What to do next?	3-20

Mass Storage Concepts 3

This chapter helps you understand some detailed concepts related to mass storage (storing files, etc., on discs). You should read this chapter to understand how to better use your mass storage devices.

Overview of Mass Storage Organization (or "What Are Files and Volumes?")

As the term "mass" suggests, mass storage devices are designed to store large quantities of data. Just how much data constitutes a large amount depends on the device itself. Most mass storage devices are capable of storing on the order of hundreds of thousands to several million items.

Mass storage devices are also called "secondary" storage media, the "primary" storage being computer memory. Because of this secondary nature, they are not generally expected to be as fast in accessing their data as primary storage.

Besides having the ability to store data, mass storage devices are capable of providing means for keeping data organized so that logical groups may be accessed systematically and efficiently. The information on a mass storage device is organized into volumes and files.

A File Is Named Collection of Data

One of the simplest ways to describe files and volumes is to use an analogy. A file is like a set papers on which information is written; we'll compare it to a tabbed chapter of a manual.

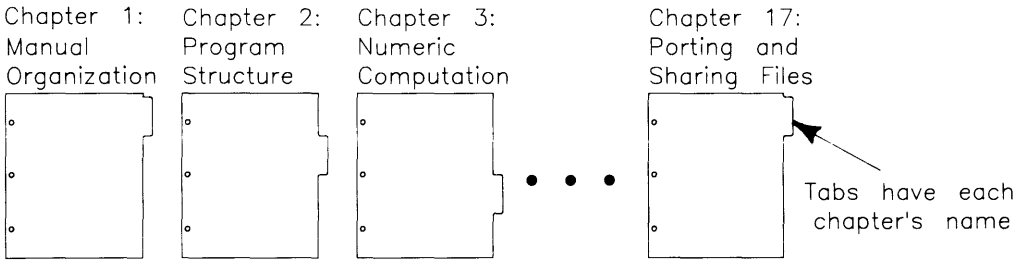


Figure 3-1. A File is Like a Tabbed Manual Chapter

Table 3-1. File Comparison

Attribute	Chapter	File
Type of information:	Printed characters, numbers, or graphics	Magnetic patterns (representing characters, numbers, dots on screen, etc.)
Access method:	Tab/chapter title	File name

Like pages in a chapter, files may contain BASIC programs or more general data—information to be used by a program, such as numbers and/or text. The name on the tab helps you locate the chapter and decide whether or not you want to look in it.

Volumes Are Collections of Files

A volume is merely a collection of files. To continue our analogy, a volume is like a binder that contains several tabbed chapters:

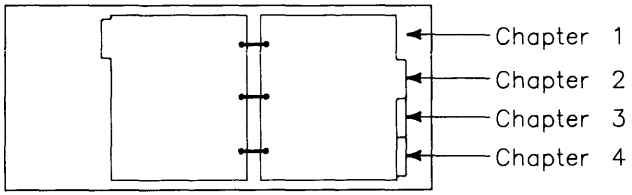


Figure 3-2. A Volume is Like a Binder Containing Tabbed Chapters

Table 3-2. Volume Comparison

Attribute	Book	Volume
Contains:	Chapters	Files
How contents listed:	Table of contents	Directory

Examples of Mass Storage Volumes

A flexible disc is typically one volume, as is a tape cartridge. Each of these types of *mass storage media* can hold several files.

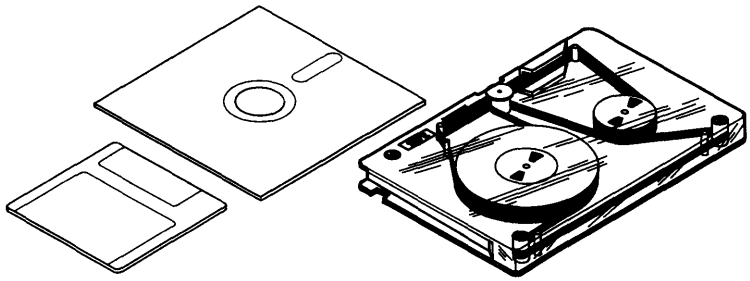


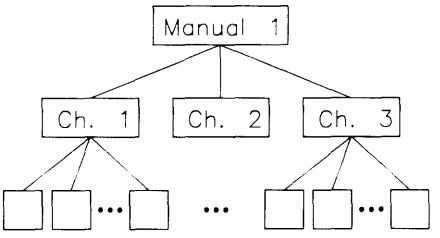
Figure 3-3. Examples of Mass Storage Volumes

Some volumes have *hierarchical* organizations. Here are two examples of hierarchies:

Textual Representation

```
BASIC  Programming  Techniques

    Chapter 1: Manual Overview
        Manual Organization
        What's in this Manual

    Chapter 2: Program Structure
        The Program Counter
        Sequence
              Linear Flow
              Halting Program Execution
              Simple Branching
        Selection
              :
              :
        Program-to-Program Communications
    Chapter 3: Numeric Computation
              :
              :
    Chapter 17: Porting and Sharing Files
```

Graphic Representation

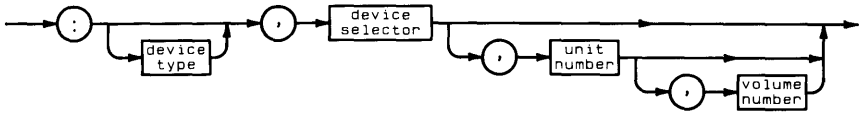


These two representations are two ways of showing the same type of hierarchical or superior/subordinate structure. BASIC also supports this hierarchical system, as described in the subsequent section called "Hierarchical Directories".

How to Specify Volumes

While it is easiest to use the default volume, there are times when you will want to use another mass storage device. In such cases, you will need to know how to specify the other volume. The way to specify mass storage devices is to use a *volume specifier*. If you read the "Verifying and Labeling Peripherals" chapter in *Installing and Maintaining the BASIC System* or followed the instructions during installation, you should already have the volume specifier of each of your mass storage devices written on a label that you placed on the device's front panel.

Here is the syntax of a *volume specifier*:



Examples:

```
:CS80,700
:,700

:HP9122,702,1
:,702,1

:INTERNAL,4
:,4,1
```

This specifier consists of the following parts:

Table 3-3. Volume Specifier Components

Component	Explanation
A colon (:)	begins the volume specifier, separating the file name from the volume specifier (note that it is required even when you are performing an operation that makes sense only for a volume—for instance, INITIALIZE.)

Table 3-3. (Continued)

Component	Explanation
Device type	identifies the mass storage device's type. Once the BASIC system determines the device type, it can also determine the capacity of the device and other information required to determine the access method for the device. Here are some examples: Device Type — Description of Mass Storage Device *if omitted* — BASIC interrogates the device for its type. `CS80` — any disc in the general class of "Command Set/80" devices (such as the HP 9122, and most other newer drives). `INTERNAL` — an internal, 5¼-inch, flexible-disc drive (Models 226 and 236 only). `HP8290X` — either an HP 82901 or 82902 5¼-inch, flexible-disc drive. `HP9895` — an HP 9895 8-inch, flexible-disc drive. For a list of all device types, see the *BASIC Language Reference* entry for MASS STORAGE IS.

Table 3-3. (Continued)

Component	Explanation
Device selector	identifies the interface's select code (4, and 7 through 31) as well as the device's primary address (HP-IB devices only). Here are some examples of device selectors: **700** specifies select code 7 and primary address 0 (note that device selectors with HP-IB addressing must contain 3 or 4 digits). **702** specifies select code 7 and primary address 2. **800** specifies select code 8 and primary address 0. **1402** specifies select code 14 and primary address 2.
Unit number	tells the system additional information about the device's unit-number setting. Many devices have hard-wired unit numbers, while others use the unit number to identify different portions of one disc. For instance, the unit number of the right drive of a 9122 is 1, while the left drive is unit 0 (note that internal drives of Model 236 computers are numbered in the opposite order).
Volume number	identifies the volume number (multi-volume hard discs only, such as HP 9133D,HP 9133H, and HP 9133L drives).

Examples

Here are several examples of volume specifiers (Note the ":," is not a typographic error. If the device type is omitted, the system will determine the device type automatically):

```
:CS80,700
```
or
```
:,700
```

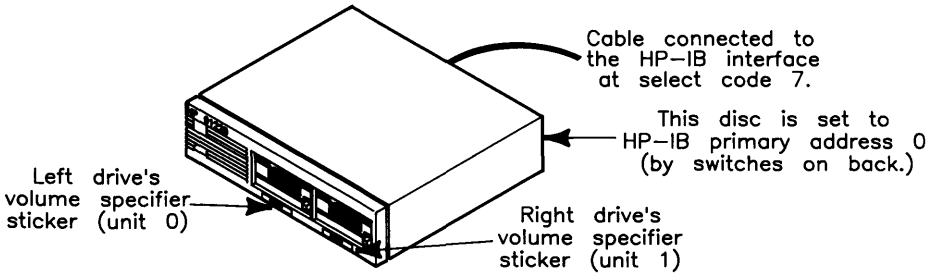


Figure 3-4. HP 9122D Volume Specifier Example

```
:HP9121,802,1
```
or
```
:HP8290x,802,1
```
or
```
:,802,1
```

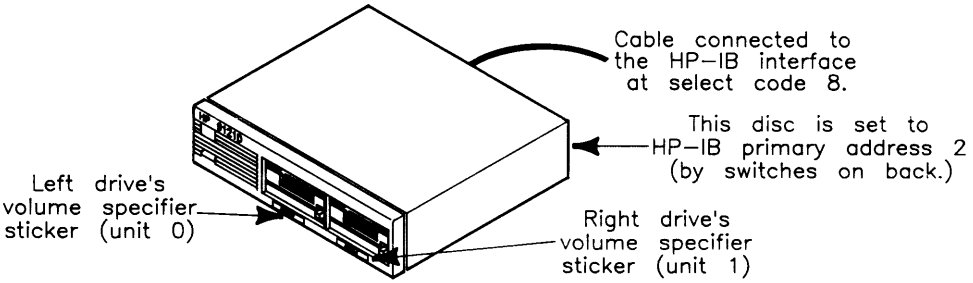


Figure 3-5. HP 82901D Volume Specifier Example

```
:CS80,1400,0,1
```
or
```
:,1400,0,1
```

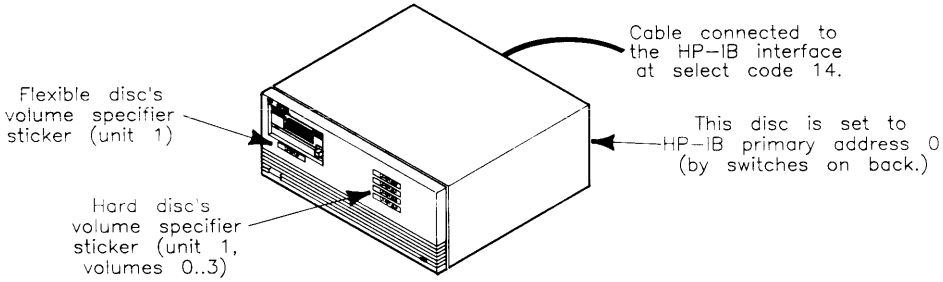


Figure 3-6. HP 9133L Volume Specifier Example

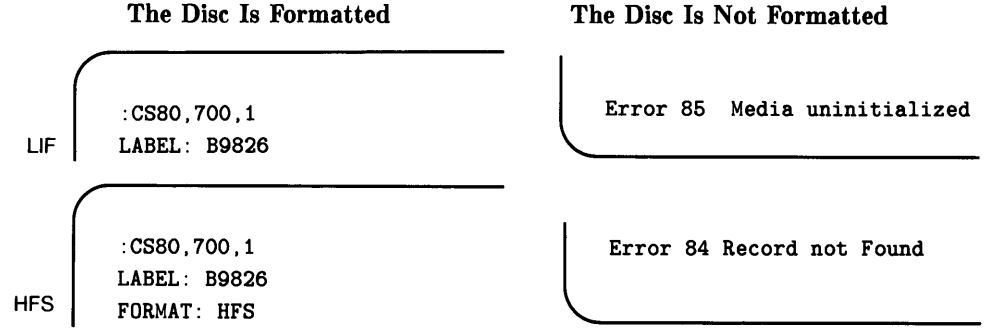
Checking a Disc's Format

To determine whether or not a disc is formatted, check to see if there is a directory on the disc. To do this, look at the catalog with the **CAT** command (you may need to specify the volume specifier if you loaded BASIC from a flexible disc):

```
CAT [Return]              Catalogs the default drive.
CAT ":,700" [Return]      Catalogs the volume :,700
```

There are two classes of results for a **CAT** operation:

- A catalog is displayed (indicating the disc is formatted)
- An error is reported (indicating the disc is not formatted; Error 1 indicates that the disc is HFS formatted but the HFS binary has not been loaded).



The top catalog listing is shown for a LIF volume, and the bottom listing is for an HFS volume.

If you are not sure which format you should use, see the next section, "Choosing a Directory Format".

Hierarchical Directories

A directory contains information about files, such as file name, size, and type. In hierarchical directory structures, a directory is itself a file, but it is used only to organize and control access to other files. This section describes the two BASIC directory formats that implement hierarchical directories:

- Hierarchical File System (HFS) format (used with HP-UX, some BASIC, and some Pascal systems)
- Shared Resource Manager (SRM) format (the disc format is actually called Structured Directory Format, or SDF, on catalog listings of these directories). Note that SRM system require an SRM controller and lets several workstations share a disc. *You cannot use HFS as a way of sharing a disc without needing a controller.*

What Is a Hierarchy?

As the word "hierarchy" suggests, hierarchical directories are arranged in "levels." A hierarchical directory may contain either files or other directories.

- A directory is "superior" to the files and directories it contains.
- A file or directory within a directory is said to be "subordinate" to the containing directory.

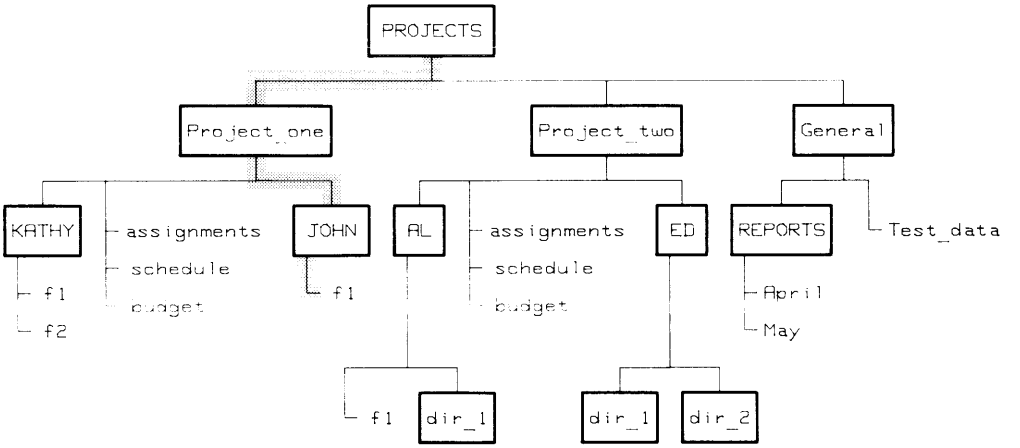


Figure 3-7. Hierarchy of Directories

In the Figure 3-4, the directory named KATHY is subordinate to the directory named Project_one because Project_one contains the information describing KATHY. The directory named PROJECTS is at level 1, the "root" level. You cannot create a directory at a higher level than the root level.

Each directory keeps information about each file or directory immediately subordinate to it, in fixed-format records. Each time a subordinate file or directory is added to a directory, one of these records is added to the directory.

Uses of the Hierarchy: An Example

Suppose you're managing several projects, each of which needs to access a shared disc. To organize the files for each project separately, you can create a directory for each project (as shown in the illustration). Within each project directory, you can have a subordinate directory for each person working on the project as well as files to be shared among all users. Each person may then construct a directory/file system for organizing files.

Because files at different locations in the directory structure can have the same file name, you can use generic file names to identify similar project functions in the different projects. At the same time, the division into separate directories isolates the projects, and thus their individual functions, from one another. For example, the file named budget in the Project_one directory is distinct from the file named budget in the Project_two directory.

Directories also limit the number of files users must deal with at any one time. For example, people working on Project_one (see illustration) need never see the files in Project_two and may, in fact, confine most of their activity to within their own directories.

To maintain security, BASIC provides the capability of protecting access to directories and files. For example, you may wish to allow only members of a project team to read that project's files. Or, you may wish to prevent other users from altering the contents of a personal file. See the "Protecting Files" section of the "Using Directories and Files" chapter.

Referring to Directories and Files in the Hierarchy

To access either a directory or a file, you must specify its location in the hierarchical directory structure. This location is specified by a list of directories, called a directory path, that you must follow to reach the desired file or directory. Directory names in the list are delimited by a slash (/).

For example, in the directory structure illustrated previously, the file specifier:

```
"/PROJECTS/Project_one/JOHN/f1"
```

defines the "directory path" to the file `f1` through its superior directories.

The directory path to a file begins at either:

- the root level, or
- the current working directory.

The current working directory is the directory specified by the most recent MASS STORAGE IS statement.

The *BASIC Language Reference* discusses the rules for specifying SRM and HFS files and directories.

Choosing a Directory Format

On the Series 200/300 BASIC system, there are three directory formats available for discs (and other mass storage media):

- Logical Interchange Format (LIF)
- Hierarchical File System (HFS)
- Structured Directory Format (SDF, used on Shared Resource Manager, or SRM, systems).

General Recommendation

The general recommendation is to:

- use HFS format with hard discs
- use LIF format with flexible discs.

> **Note**
>
> If you wish to share a disc between several workstations, you need an SRM system. HFS or LIF are only disc formats and not systems with a controller like SRM.

To show you why these are recommended, see the characteristics of each format in the following table.

Criteria for Choosing a Directory Format

Here are the criteria in choosing between LIF and HFS directory formats.

Feature	HFS	LIF	For More Info
Directory structure	Hierarchical (multi-directory) structure.	Single directory on each volume.	See preceding examples, and the "Using Files and Directories" chapter.
Multiple systems on same volume	HP-UX, BASIC, and Pascal systems can share a disc.	BASIC and Pascal can share a disc.	See *Installing and Maintaining the BASIC System*
File compatibility	"Text" files are the interchange method.	"ASCII" files are the interchange method.	See the "Porting and Sharing Files" chapter of *BASIC Programming Techniques.*
Extensible files	Files are extensible (when a file would otherwise overflow, the system automatically adds space to it)	File length is fixed.	See "Data Storage and Retrieval" chapter of *BASIC Programming Techniques.*
Access times	Generally slower than LIF.	Generally faster than HFS.	For more exact data, you can use the benchmarking methods shown in the "Efficient Use of the Computer's Resources" chapter of *BASIC Programming Techniques.*
TRANSFER	Not really implemented as a background process.	True background process and high data rates.	See the "Advanced Transfer Techniques" chapter of *BASIC Programming Techniques.*
Overhead required	Requires slightly more overhead than LIF.	Requires slightly less overhead than HFS.	See the next section for examples.

Examples of HFS Overhead

This section shows some figures regarding how much of a disc is used as "overhead" on an HFS volume. (Overhead is the space on the disc required to store just the directory format and not any data files.)

Disc Size	Approximate Overhead
single-sided 3½-inch, and 5¼-inch flexible discs	44% (256-byte sectors)
double-sided 3½-inch flexible discs	22% (256-byte sectors) 18% (1024-byte sectors)
55 Megabyte hard disc	6%
130 Megabyte hard disc	6%

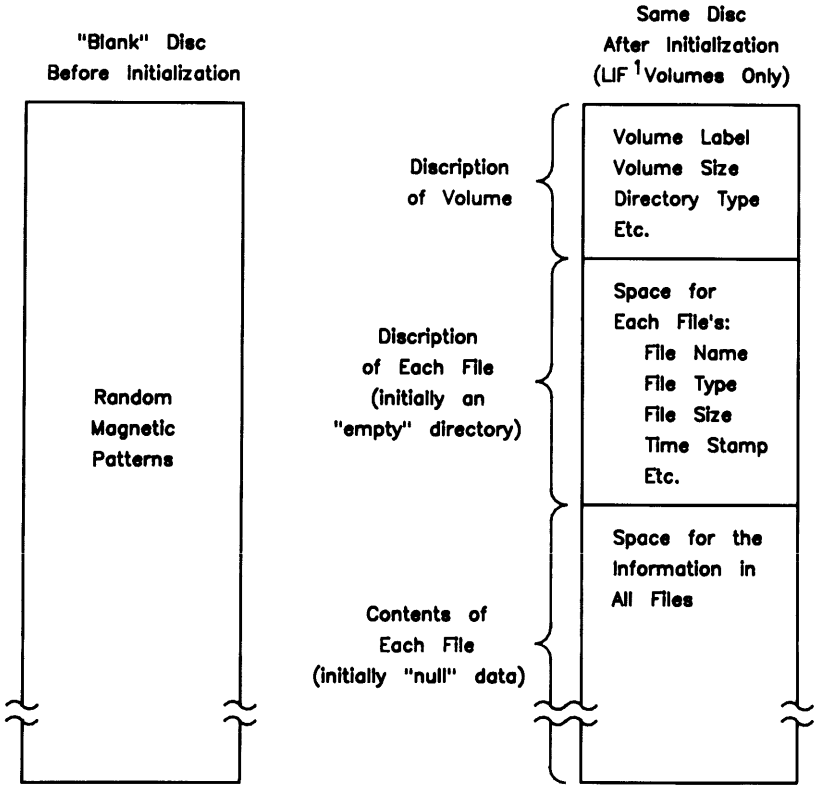
In addition, you should keep the disc less than 90% full for optimum performance.

These figures show only the "extremes" of the spectrum; however, they do show that:

- The overhead for larger hard discs is fairly constant at about 6%.
- Overhead for flexible discs may be large.

What Is Initialization?

When a "blank" disc is shipped from the factory, there are no files on it. In fact, there is not even an empty directory on it. You will need to put a directory on the disc before using it. (Some computer systems and documentation call this "formatting" a disc.) The "System Disc Utility" (DISC_UTIL) or the INITIALIZE statement will do this for you.



[1] HFS & SRM volumes have different structures, since they can have multiple directories on each volume.

Bad Sector Scan

Initialization writes "null" data onto the disc, and then reads the data to verify that it was written correctly. If the data was stored properly in a particular "sector" of the disc, then that sector is considered to be functional. However, if what is read back from the disc is not what was written onto it, then the sector is considered "bad" and is marked accordingly—or "spared" (and is not used subsequently).

Note

Note that initialization is also a good thing to do periodically to remove "bad" sectors from use. **(But remember to first make a back-up copy of the information on the disc, since all files will be overwritten!)**

Disc Sectors

Data on mass storage devices is written in "sectors". A sector is a set of contiguous bytes (eight bits, or binary units) on the disc (on the same track). A sector is the smallest unit of data that can be written and read. If you want to change one byte on a disc, you must read the entire sector, change the byte, then write the entire sector to the appropriate location.

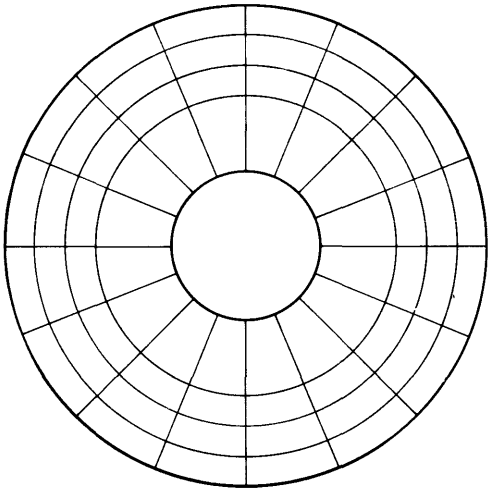


Figure 3-8. Discs are Accessed by Sectors

Sector/Volume Size Option

With this BASIC system, sector sizes of 256 and 1024 bytes are available. The "System Disc Utility" uses the default sector/volume-size option of 1, which specifies 256-byte sectors. If you want a non-default sector size, then you must use the INITIALIZE statement.

Disc Interleave

Interleaving a disc causes the sectors on the disc to be numbered according to a specific interval. For instance, an interleave factor of 1 causes sectors to be numbered consecutively. A factor of 2, on the other hand, causes the system to skip every other sector. The following drawing illustrates these two interleave factors.

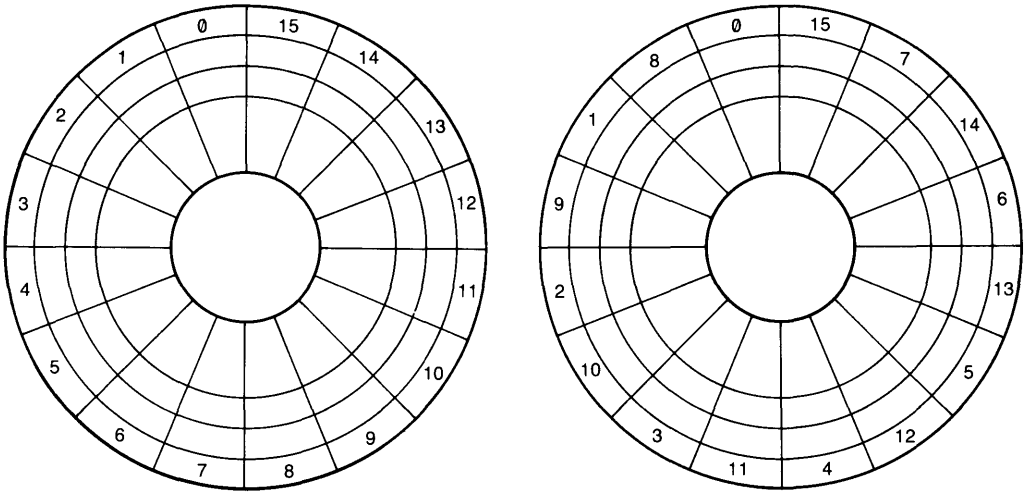


Figure 3-9. Examples of Disc Interleave

The BASIC system numbers each sector on the disc using the same interleave factor; that is, each file cannot have a separate interleave factor.

The purpose of disc interleave is to increase data-transfer rates, as demonstrated in the following example. Suppose that we are entering data from a spinning disc; suppose also that the data have formats which are different from the computer's internal data formats. Consequently, after each item is read, the computer must change the data from the disc's format to the computer's internal format. And in the meantime, the disc is still spinning.

If the processing of all items in a sector takes more time than it takes for the next sector to pass under the read/write head, then the system must wait one full revolution of the disc until the next sector again passes under the head. By interleaving a disc, you can provide time for this processing, thus not making the system wait as long until the next sector passes under the read/write head.

You can choose the interleave factor when you initialize a disc using the INITIALIZE statement (but not when you initialize it using the "System Disc Utility"—the DISC_UTIL program). See the Installing section of *Installing and Maintaining the BASIC System.*

What to Do Next

Task/Topic	Chapter
How to run BASIC programs	"Loading and Running Programs"
How to store files	"Using Files and Directories"
How to edit programs	"Editing and Storing Programs"

Steps in Installation	Where to Go For Details
A brief look at loading and running programs	4-1
A closer look at loading programs	4-4
Using LOAD	4-4
Using GET	4-5
A closer look at running programs	4-7
What to do next?	4-14

Loading and Running Programs 4

In this chapter, you will learn how to load and run programs written in BASIC.

A Brief Look at Loading and Running Programs

This section gives you a brief overview of loading and running BASIC programs. To load and then run a program, follow this procedure:

1. If the program is on a flexible disc, insert the disc containing the program into one of your disc drives. The best choice is the "default drive," since you will not have to include the drive's *volume specifier* when you use this device. If the program is on your hard disc, continue with Step 2.

2. Execute the CAT statement to make sure your program is on the disc (include the volume specifier written on the sticker on your disc drive if the program is on a disc which is not the default drive; for example `:,702,1`).

   `CAT` [Return]

   - if you see the name of your file in the catalog listing, you are ready to load and run it

   - if you don't see your file in the catalog listing, then remove the disc and insert the one that contains your program. Use CAT again to verify that the program is on the disc.

3. Use the LOAD or GET command to load the program into computer memory:

   `LOAD "`*filename*`"` if the file is type PROG (refer to the CAT listing)

   `GET "`*filename*`"` if the file is type ASCII or HP-UX (refer to the CAT listing).

Examples

```
:CS80,700
VOLUME LABEL: B9836
FILE NAME PRO TYPE  REC/FILE BYTE/REC   ADDRESS

MY_PROG       PROG       14      256        16
VISI_TOOL     ASCII      29      256        30
GRAPH         BIN       171      256        59
GRAPHX        BIN       108      256       230
```

Here are some statements to load PROG and ASCII programs. Since `MY_PROG` is type PROG, you would use:

```
LOAD "MY_PROG"
```

to load it. For the file `VISI_TOOL`, you would use:

```
GET "VISI_TOOL"
```

assuming of course the file is in the current directory. If we used a file located on a disc **not** in the default drive, a volume specifier would be required. For example, if after labeling the discs following the procedures in "Verifying and Labeling Peripherals" in the *Installing and Maintaining the BASIC System* manual, we had a PROG file named `OUR_PROG` on a disc with volume specifier `CS80,700,1`, here is the statement to load the file:

```
LOAD "OUR_PROG:,700,1"
```

Note that we don't need to include the `CS80`, and we follow the file name immediately with a colon (:) and a comma (,) then the numbers.

4. To run the program you just loaded, type:

   `RUN` [Return]

   On 98203 keyboards, you can also use the [RUN] key. On ITF keyboards, [f3] in the System menu, and User 1 and 2 menus serves the same purpose. (If key labels are not currently displayed, then execute KEY LABELS ON or press the [Menu] key to turn them on.)

Note about Software Security

Some software is "secured" against being run without proper authorization which is usually accomplished by the software requiring a special "codeword" that is somehow related to:

- your machine's serial number (stored in permanent memory)
- a serial number stored in an optional HP 46084 ID Module

If the program prompts you to enter a codeword, you will need to get it from the **software vendor**.

A Closer Look at Loading Programs

There are two statements used in BASIC for retrieving programs from mass storage:

- LOAD—retrieves programs stored in a PROG file (using STORE)
- GET—retrieves programs stored in an ASCII or HP-UX file (using SAVE).

These statements can be executed from the keyboard as commands or included in a program. When executed as commands, they are used to bring a program into the computer's memory so it can be edited or run. When included within a program, they are used to link together the segments of large programs.

Using LOAD

The LOAD command brings in programs from a PROG file, with the option of beginning program execution at a specified line. It clears any existing program from the computer's memory and loads the contents of a PROG file. For example, the command:

```
LOAD "CANNON"
```

clears the program memory and brings in the contents of the PROG file called "CANNON".

Possible Errors

- If the file is not a PROG file, the LOAD is not performed and error 58 (improper file type) is reported.
- If the file is not found on the volume, error 56 (file not found) is reported.
- If any lines require a version of BASIC different from the one currently installed, those lines cannot be listed, edited, or executed. However, the LOAD operation proceeds without error.

Using LOAD to Specify Run Line

The LOAD command can also specify that program execution is to begin. This is done by adding a line identifier. For example, the command:

```
LOAD "STONE",10
```

causes the computer to load the program in file "STONE" and begin execution at line 10. The line identifier may be a label or a line number, but it must identify a line in the main program segment (not in a subprogram or user-defined function). See the "Program Flow" chapter of *BASIC Programming Techniques* for further information.

The LOAD command cannot be used to bring in arbitrary program segments or append to a main program like GET can. Subprogram segments can be appended using LOADSUB, as described in the "Subprograms" chapter of *BASIC Programming Techniques.*

Using GET

The GET command brings in programs or program segments from an ASCII or HP-UX file, with the options of appending them to an existing program and/or beginning program execution at a specified line.

GET with Automatic Program Clearing

To clear any existing program from the computer's memory and bring in the contents of an ASCII or HP-UX file, the command is simply the keyword GET followed by the file name. For example, this command:

```
GET "FORMULA"
```

clears any BASIC program currently in memory, and brings in the contents of the ASCII or HP-UX file called "FORMULA"—assuming that the file contains valid program lines.

Possible Errors

- If the first line does not start with a valid line number, the GET is not performed and `error 68` (syntax error during GET) is reported.
- If the file is not an ASCII or HP-UX file, the GET is not performed and error 58 (improper file type) is reported.
- If the file is not found on the volume, error 56 (file not found) is reported.

Assuming the file contains valid program lines that were placed in the file by a SAVE operation, and their line numbers are still valid after any renumbering that is specified, the lines will be entered into program memory. If there is a syntax error in any of the program lines in the file, the lines in error are turned into comments, error 68 is reported, and the syntax error message is sent to the system printer. This might happen if the program was written and saved on a computer that had a version of BASIC different from the one being used for the GET operation. This may also happen when a "language extension" binary is required for using the keyword, but the binary is not currently loaded into memory.

Using GET to Specify Run Line

GET can also specify that program execution is to begin. This is done by adding two line identifiers:

- the first line number specifies the placement and renumbering
- the second line number specifies the line at which execution is to begin.

For example, assume there is no program in memory and an ASCII file "RATES" contains valid program lines. A typical command to bring the contents of this file into memory and begin execution at the line 10 is:

```
GET "RATES",10,10
```

If there is **already** a program in memory, an append and run is allowed. For example:

```
GET "RATES",250,100
```

specifies that any existing lines from 250 to the end of the program in memory are to be deleted, the contents of file "RATES" is to be renumbered and appended to the program in memory beginning at line 250, and then normal program execution is to begin at line 100. Although any combination of line identifiers is allowed, the line specified as the start of execution must be in the main program segment (not in a SUB or user-defined function). Execution will not begin if there was an error during the GET operation. For further information about this use of GET, see the "Chaining Programs" section of the "Program Flow" chapter in *BASIC Programming Techniques.*

Using GET to Append

You can also use GET to append the contents of an ASCII file to an existing program. See the "Editing and Storing Programs" chapter or the *BASIC Language Reference* for details.

A Closer Look at Running Programs

There are two ways to start a program running:

- press the [RUN] key ([f3] in the System and some User menus of ITF keyboards)
- execute a RUN command.

You can include a line identifier in a RUN command to indicate where program execution is to begin. For example,

```
RUN 200
```

or

```
RUN Line_id
```

Prerun

Prerun is automatically performed by BASIC when you execute RUN or press the [RUN] key. There are several reasons for the prerun.

- **To reserve sufficient memory** for all the variables in the program (except those that are ALLOCATEd). This includes all variables in COM statements, those declared in DIM, REAL, and INTEGER statements, and all implicitly declared variables. The "Numeric Computation" chapter of *BASIC Programming Techniques* explains the declaration of numeric variables; the "String Manipulations" chapter covers the dimensioning of string variables; and the "Subprograms" chapter describes COM.
- **To locate all the context boundaries.** These are defined by the END, SUB, SUBEND, DEF FN, and FNEND statements.
- **To ensure correct interaction between lines.** The "Editing and Storing Programs" chapter explains that BASIC checks for syntax errors before it stores a program line. Although this is true, there are some errors that can't be detected by looking at a single line. For example, a program line that uses properly placed subscripts can appear to be correct when it is stored. However, if that line references three dimensions in an array that had previously been declared to have only two dimensions, it is in error. To detect an error of that kind, the computer needs to "look at" the entire program to see all the dimension statements as well as the variables used in each line. Some other examples of this kind of error are specifying a GOTO or GOSUB to a line that does not exist or improper matching of statements like FOR...NEXT and IF...END IF. At prerun, BASIC also "links" line identifiers (and line numbers) to all references to them, so that program execution will be faster.

Normal Program Execution

The term "program execution" is used to describe the process of the BASIC system completing the tasks "described" by a program. The steps that the system performs during program execution are summarized below.

1. Determine which program line is to be acted on next.
2. Identify the statement that follows the line number and label (if any) on that line.
3. If the statement has a run-time action, perform the action described in the statement.
4. Repeat steps 1 through 4 until instructed to pause or stop (such as by an END, STOP, or PAUSE statement, or by a [RESET] from the keyboard).

The continuing process of determining which line is to be executed next is discussed in detail in the chapter called "Program Flow" in *BASIC Programming Techniques*. The RUN command determines which line is acted on first. Executing **RUN** with no parameters, or pressing the [RUN] key, causes the execution process to begin at the first (lowest-numbered) line of the main program. Execution can be started anywhere in the main program by using the RUN command with a line identifier. For example, the following command causes execution to begin at line 220, if there is such a line.

```
RUN 220
```

If there is no line 220 in the main program, execution begins with the line whose number is closest to and greater than 220.

The line identifier can also be a label. For example, the following command causes execution to begin with the line labeled "Spot_run".

```
RUN Spot_run
```

If there is no such label, an error results.

Also, if a starting line is specified, that line must be in the main program. An error 3 (line not found in current context) results if you attempt to start a program in a user-defined function or subprogram. Even if the starting point is correctly specified, be alert to the effects of starting a program in the middle. Skipping over a section of the program may result in null values for some of the variables. Although it is legal to start in the middle of a subroutine, an error is generated when the RETURN statement is executed.

Live Keyboard

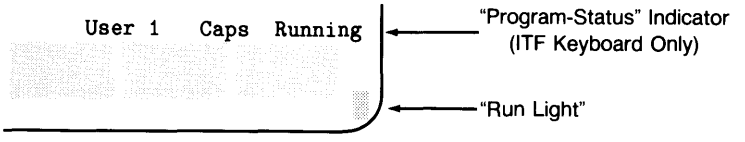
When a program is running, the keyboard is still active. Commands can be executed, variables can be inspected and changed, and the state of the computer can be changed. The term "live keyboard" is used when talking about commands that are executed during a running program. One of the principal uses for live keyboard commands is the troubleshooting and debugging of programs in the development stage, as discussed in the "Debugging Programs" chapter of *BASIC Programming Techniques*. See "Introduction to the System" for tables showing how to pause, stop and continue a program.

Example of Controlling Program Execution

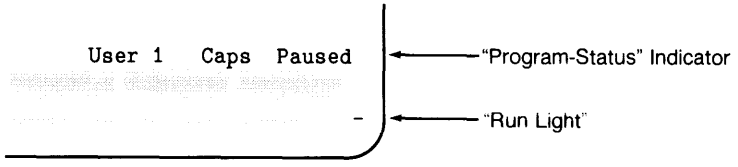
To demonstrate some of the interaction between a program and the keyboard, enter the following simple program.

```
10  DISP "Next command?"
20  X=0
30  PRINT X;
40  X=X+1
50  WAIT .1
60  GOTO 30
70  END
```

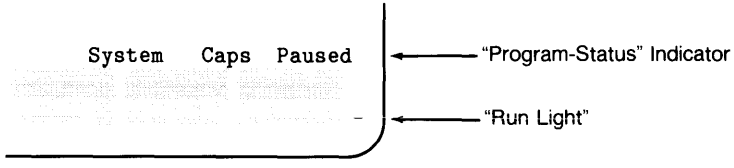
1. After you have entered the program, execute RUN and observe the CRT. Notice that the DISP message appears in the display line, the printout area fills with a sequence of numbers, and the run light indicates that a program is running.



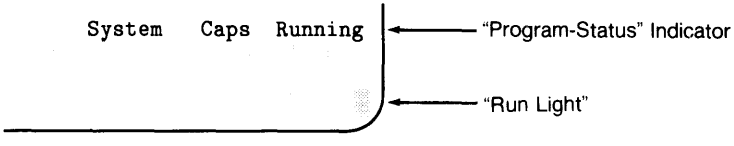
2. Press [PAUSE] ([Stop] on an ITF keyboard). The printout of numbers stops, and all the data on the CRT remains unchanged. The run light now indicates that the program is paused and can be continued. The program line that appears at the bottom of the CRT is the next line that will be executed when program execution resumes.



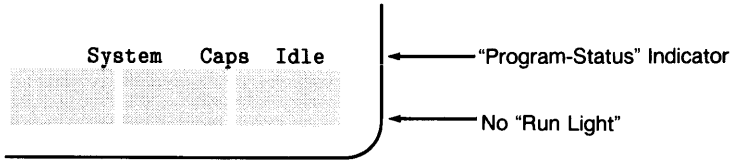
3. Press [STEP] ([f1] on an ITF keyboard) a few times. The program is executed one line at a time, as indicated by the lines changing at the bottom of the CRT. Notice that the program is still paused and continuable after each press of the [STEP] key. The [STEP] key can be a great help when you are trying to find certain kinds of problems. The "Debugging Programs" chapter of *BASIC Programming Techniques* gives the details of this and other debugging tools.



4. Press [CONTINUE] ([f2] on an ITF keyboard). The printout on the CRT resumes with the next number in the sequence. The run light again indicates that a program is running.

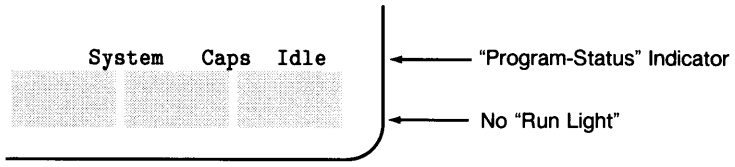


5. Press [Shift]-[Stop] (ITF keyboard) or [STOP]. The printout of numbers stops, and all the data on the CRT remains unchanged. However, the run light is off, indicating a stopped condition.



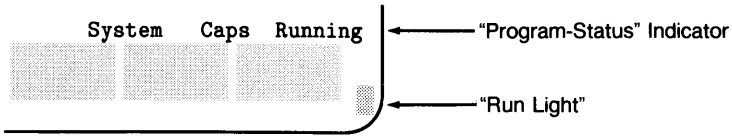
6. Press [CONTINUE] ([f2] on an ITF keyboard). An error results, because a stopped program cannot be continued.

```
Error 122: Program not continuable
```

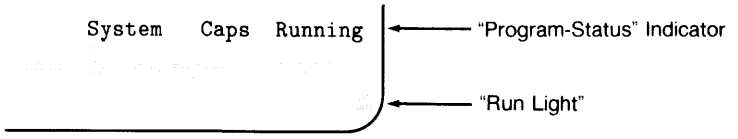


7. Press [RUN] ([f3] on an ITF keyboard).

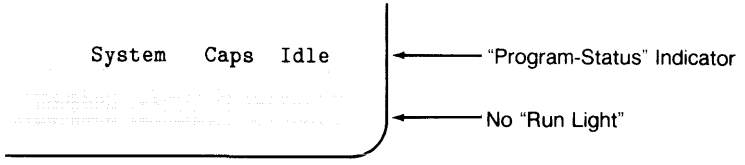
   The program runs again, but the number sequence has restarted from the beginning, not from the next number in the sequence. RUN causes the program to restart, not resume.



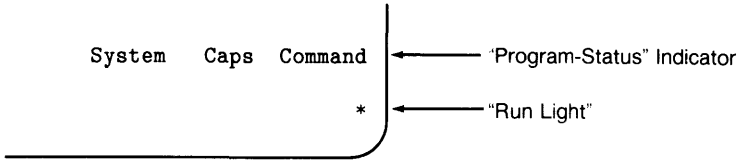
8. Type `X=1` and press [Return]. Notice that the numbers being printed start over with "1". The live keyboard was used to change the value of "X", and the program used the new value from the keyboard.



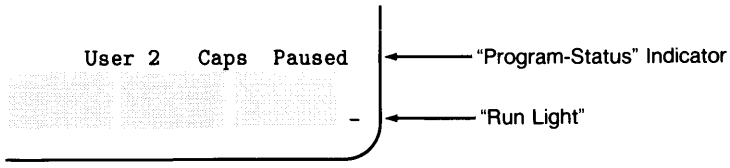
9. Press [Reset]. The program stops and the data remains in the printout area, but the display line is cleared and the message **`BASIC Reset`** appears at the bottom of the CRT. Although the clearing of the display line seems like a minor effect, it indicates an important point. [Reset] and [Stop] have different effects on interfaces and peripheral devices. This aspect of [Reset] is summarized in the "Reset Tables" in the "Useful Tables" appendix of the *BASIC Language Reference* and is discussed fully in the *BASIC Interfacing Techniques* manual.



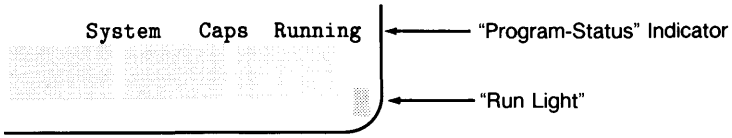
10. Press [RUN] ([f3] on an ITF keyboard). Then type `WAIT 5` and press [Return]. Notice that the run light changes to indicate that a keyboard command is being executed and the printout is delayed for five seconds while the live keyboard command is processed. Actually, the run light changed when the `X=1` command was executed in step 8, but it may have happened so fast that you didn't see it.



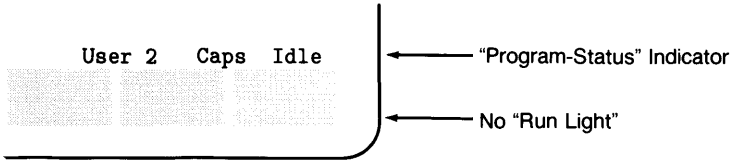
11. Press [PAUSE] ([Stop] on an ITF keyboard) and then type **EDIT** and press [Return]. The display on the CRT changes to show the program. The line you were editing last appears in the current-line position. Notice that the run light is still visible in the lower right-hand corner and it indicates that the program is paused.



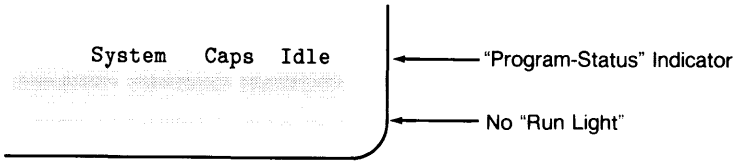
12. Press [CONTINUE] ([f2] on an ITF keyboard). The CRT returns to normal mode, and the printout of numbers continues in sequence. However, the previous data on the display was lost when the CRT was used for EDIT mode.



13. Press [PAUSE] ([Stop] on an ITF keyboard). Then type **EDIT 50** and press [Return]. The CRT changes to EDIT mode and the program appears again. This time, line 50 is in the current-line position. Notice that the run light indicates that the program is paused. Change line 50 to **WAIT .2** and press [Return]. The new line 50 is entered, but the run light changes. Editing the program caused it to move from the paused state to the stopped state.



14. Press [CONTINUE] ([f2] on an ITF keyboard). An error results. As mentioned earlier, a program can be viewed while it is paused, but it cannot be changed. Once any program line has been changed, the program is no longer paused, and [CONTINUE] is not allowed.



This demonstration covers most of the highlights of live keyboard, program states, and the run light. The "waiting for input" indication can be seen when using the INPUT and LINPUT statements described in the chapter of *BASIC Programming Techniques* called "Communicating with the Operator". The "paused with I/O completing" indication is not described in this manual. It is a special state that results from the use of TRANSFER and is discussed in the *BASIC Interfacing Techniques* manual.

What to Do Next

Task/Topic	Chapter/Section
Learn how to use and manage files and directories	"Using files and Directories"
Learn about each key on your keyboard	"Keyboard Reference"
Learn about editing and storing programs	"Editing and Storing Programs"
Learn about utilities available with the BASIC system	"BASIC Utilities Library", *Installing and Maintaining the BASIC Manual*

Task/Topic	Where to Go for Details
Finding and specifying files	5-1
Is the file on the default volume?	5-1
Changing the default volume and current	5-6
Creating and using hierarchical directories	5-7
A closer look at file catalogs	5-13
LIF catalogs	5-14
HFS catalogs	5-15
SRM catalogs	5-17
General file management operations	5-21
Closed vs. open files and hierarchical directories	5-21
Protecting files (PROTECT, PERMIT, LOCK/UNLOCK)	5-21
Copying files and volumes (COPY)	5-32
Renaming files (RENAME)	5-34
Purging files (PURGE)	5-35
Volume labels (LIF and HFS volumes only)	5-39
Enabling checkread verification (CHECKREAD)	5-40
What to do next?	5-41

Using Directories and Files 5

The main use of a computer is to run programs that process some sort of data. Both programs and data can be stored in files, which reside in mass storage volumes. The files in a particular volume are listed (and described) in directories. This chapter describes general management of directories and files.

Finding and Specifying Files

A file is a collection of information accessed by a single name. A volume is a collection of files. (For an introductory explanation of volumes and files, see the "Mass Storage Concepts" chapter of this manual.) This section shows how to locate and specify BASIC files.

Is the File on the Default Volume?

Use the CAT (catalog) statement to determine the names of the files on a volume.

If you do not specify a volume, the default volume will be assumed. (Examples of specifying a volume other than the default volume are presented subsequently.)

```
CAT [Return]
```

You should get a listing similar to this, which you can visually scan for the presence of the desired file(s).

```
:CS80,700
VOLUME LABEL: B9836
FILE NAME PRO TYPE  REC/FILE BYTE/REC   ADDRESS

MyProg        PROG        14      256        16
VisiComp      ASCII       29      256        30
GRAPH         BIN        171      256        59
GRAPHX        BIN        108      256       230
```

Names of the files on the volume

Sending Catalogs to External Printers

The CAT statement normally directs its output to the current PRINTER IS device (the default is the CRT display). The CAT statement can also direct the catalog to a specified device, as shown in the following examples:

```
CAT TO #701
CAT TO #External_prtr
CAT TO #Device_selector
```

The parameter following the # is known as a device selector and is further described in the chapter of *BASIC Programming Techniques* called "Using a Printer" and in the Glossary of the *BASIC Language Reference.*

Specifying Files on the Default Volume

To specify a file on the **default volume**, merely use its name. Here are examples of accessing files from the current default volume:

```
LOAD "MY_PROG"
GET  "PROG_ASCII"
PURGE "A_FILE"
```

Specifying Files Not on the Default Volume

If the file is *not* on the default volume or not in the current working directory (if using a hierarchical directory), then you will need to do one of the following things:

- Identify the volume in the file specifier. For instance:

```
LOAD "MY_PROG:,700"
```

- Change the current working directory and/or default volume. For instance:

```
MASS STORAGE IS "/USERS/MARK"
LOAD "MY_PROG"

MASS STORAGE IS ":,700"
LOAD "MY_PROG"
```

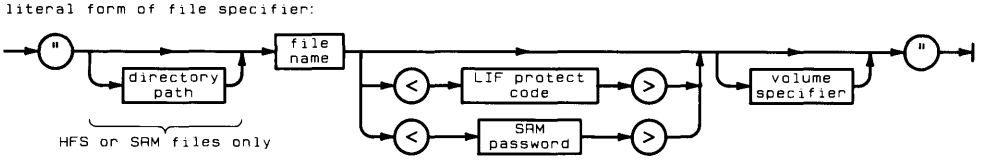
Let's first take a closer look at the alternative of specifying the volume with the file name.

Directory, File, and Volume Specifiers

Files can be uniquely identified by specifying the following information:

- The directory in which the file resides (if it is in a hierarchical directory structure).
- The file's name. (If a LIF or SRM file is currently "protected", you will need to include the "protect code" or "password" with the file name. If an HFS file is "protected", you will need to make sure you have the correct access permission. See the subsequent section of this chapter called "Protecting Files" for details.)
- The volume on which it resides. (If the file is on the default volume, then you don't need to specify the volume; BASIC *assumes* the file is on the *default volume* when you don't specify one.)

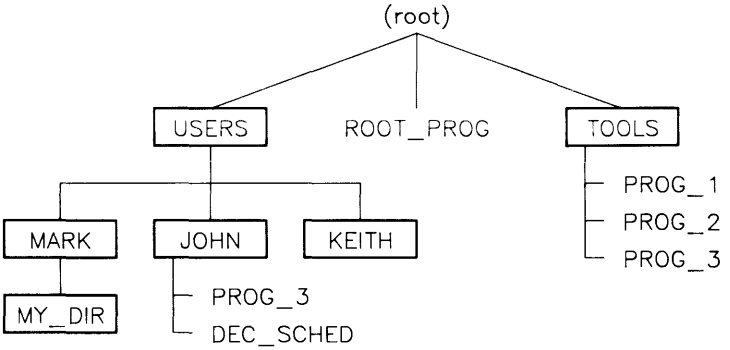
Here is a syntax drawing of the components of a file specifier:



Files Not in the Current Working Directory (Hierarchical Directories Only)

If you want to specify a file (in a hierarchical directory) but it is not in the *current working directory*, then you must include this information in the file specifier.

Here is an example directory structure that will be used in the subsequent examples in this section. None of the examples change the default volume; all remain in the directory structure of the current default volume. (The next section shows examples of changing the current default volume.)



For instance, to catalog the root directory of the current default volume, you type:

```
CAT "/"
```

To load the file ROOT_PROG from the root directory, you type:

```
LOAD "/ROOT_PROG"
```

Next, to catalog the "USERS" directory:

```
CAT "/USERS"
```

which gives you a list of the files MARK, JOHN and KEITH. To load the file "PROG_3" from the directory named "JOHN" (subordinate to the directory named "USERS" in the root):

```
LOAD "/USERS/JOHN/PROG_3"
```

This example catalogs the directory which is superior to the current working directory:

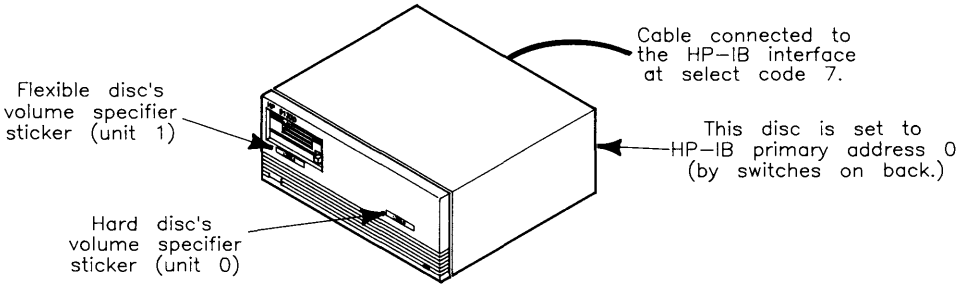
```
CAT ".."
```

If you were located in the MARK directory, for example, this command would catalog the USERS directory (located above MARK in the above diagram).

Files Not on the Default Volume

If the file is not on the *default volume*, then you must include the volume specifier. The *volume specifier* is the information on the label which you should have affixed to each of your mass storage volumes during installation. (If not, see the "Verifying and Labeling Peripherals" chapter in *Installing and Maintaining the BASIC System* manual.

Here is a pictorial description of a typical hardware configuration (a device with both a hard and a flexible disc drive):



The following LOAD statement specifies the file named "MY_PROG", located on the volume connected to the computer through the HP-IB interface at select code 7, with primary address 0, and unit number 1:

```
LOAD "MY_PROG:CS80,700,1"
```

or

```
LOAD "MY_PROG:,700,1"
```

The following CAT statement catalogs the hard-disc volume in the above example; it is at the same select code and primary address, but it has a unit number of 0 (which is the default when the unit number is not specified):

```
CAT ":,700"
```

This example catalogs another volume (at select code 8, primary address 2, unit number 1):

```
CAT ":,802,1"
```

Changing the Default Volume and Current Working Directory

The following statement sets the default mass storage to an HP 9133 drive at interface select code 7 with primary address 0; unit number 1 specifies the flexible-disc drive:

```
MASS STORAGE IS ":CS80,700,1"
 or
MSI ":,700,1"
```

If the volume has a hierarchical directory structure (HFS volumes), then you may also specify a *current working directory*:

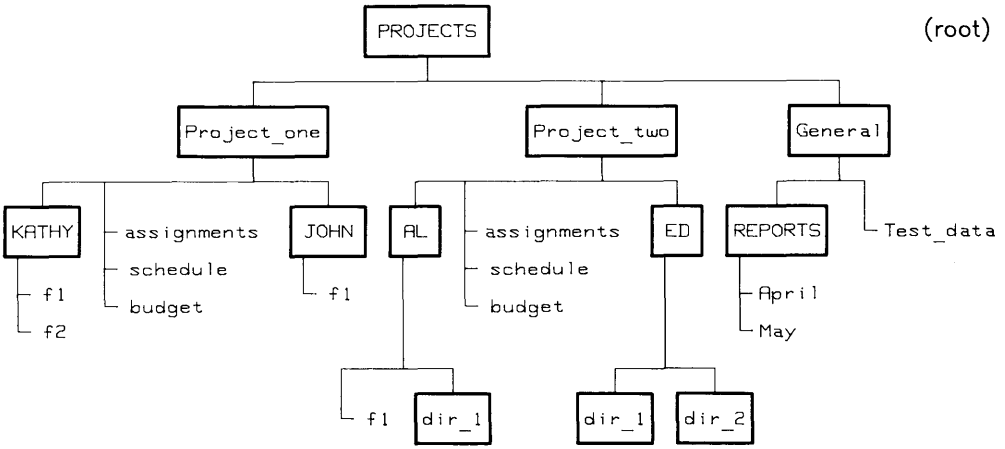
```
MSI "/USERS/MARK:CS80,700,1"
 or
MSI "/USERS/MARK:,700,1"
 or
MSI "/USERS/MARK"    If the default volume is ":,700,1"
```

Creating and Using Hierarchical Directories

Directories contain information about files on a volume. If you are on a hierarchical-directory volume (such as HFS or SRM), directories also have additional capabilities. This section shows how to create and access hierarchical directories.

Example Hierarchy

Examples in the following paragraphs refer to the directory structure shown in the illustration below.



The boxed names signify directories, while the unenclosed names signify files.

Changing the Default Volume

First, it will be helpful to change the default volume to the "root" directory of your hierarchical volume by executing something like this:

```
MSI ":,700"        or
MSI ":,21,0"
```

You do not need to specify a "/", since the directory path is assumed to begin at the root directory when the volume specifier is included.

Adding Another Directory

This statement creates a directory named **CHARLIE** in this directory structure:

```
CREATE DIR "/PROJECTS/Project_one/CHARLIE"
```

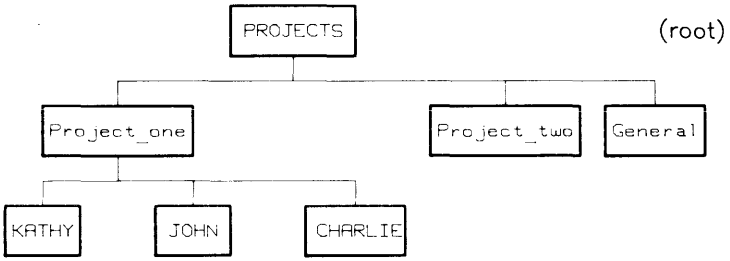
The leading slash indicates that the directory path begins at the root of the hierarchical directory structure. Each subordinate directory is listed from left to right, separated by slashes (**PROJECTS/Project_one/**). The directory being created is listed after the directory path (**CHARLIE**).

You could accomplish the same results with the following statements:

```
MSI "/PROJECTS/Project_one"

CREATE DIR "CHARLIE"
```

Note that in both of the preceding examples, the leading "/" was unnecessary since the current working directory was already the root directory.

This statement would place your newly created directory into the directory structure as shown below.



Creating Files and Other Directories Under a Directory

To create files subordinate to a new directory, you may either establish the new directory as the working directory or specify the directory path to that directory. Assuming your current working directory is the root, you could type:

```
MSI "PROJECTS/Project_one/CHARLIE"
```

to move into the directory **CHARLIE**.

You could verify the new working directory with a catalog listing by typing:

```
CAT
```

On a computer whose screen supports an 80-character line width, the resulting listing would look something like this (this example is from an SRM):

```
PROJECTS/Project_one/CHARLIE:CS80, 700, 0, 0
LABEL:     Disc1
FORMAT:    SDF
AVAILABLE SPACE:        54096
                        SYS  FILE   NUMBER   RECORD     MODIFIED  PUB OPEN
FILE NAME          LEV TYPE  TYPE  RECORDS   LENGTH DATE     TIME ACC STAT
================== === ==== ===== ======== ======== ============ === ====
```

To create an ASCII file within **CHARLIE**, which is named **ASCII_1** and is initially to contain 100 records, execute this statement:

```
CREATE ASCII "ASCII_1",100
```

To create a BDAT file within **CHARLIE**, which is named **BDAT_1** and is initially to contain 25 records, execute this statement:

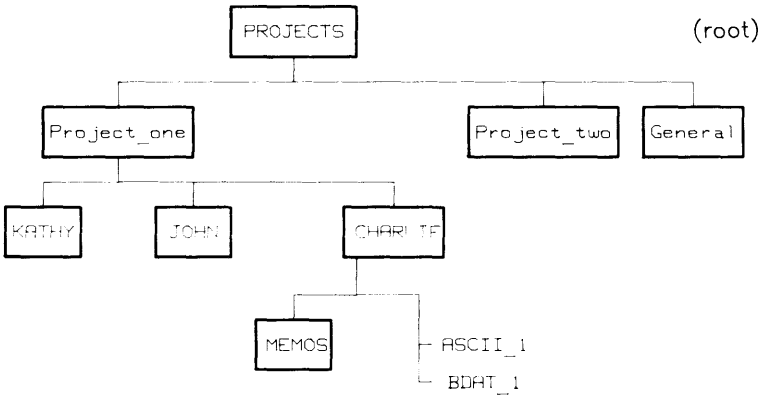
```
CREATE BDAT "BDAT_1",25
```

(When no record size is specified in the CREATE BDAT statement, the default 256-byte record size is assumed.)

To create another directory within **CHARLIE** called **MEMOS**, execute this statement:

```
CREATE DIR "MEMOS"
```

The additions would make the directory structure look like this:



The simplest form of the CAT statement:

```
CAT
```

lists the contents of the current working directory because no directory is specifically identified. If no directory name is shown in the directory header, the current working directory is the root.

If you wanted to list the contents of **CHARLIE**, but your current working directory was **not** **CHARLIE**, you could:

- Designate **CHARLIE** as the working directory with the MSI statement, then use the CAT statement's "short form." For example:

  ```
  MSI "PROJECTS/Project_one/CHARLIE:REMOTE"
  ```
  (for an SRM disc)

  or

  ```
  MSI "PROJECTS/Project_one/CHARLIE:,700"
  ```
  (for an HFS disc)

  ```
  CAT
  ```

- In the CAT statement, specify the entire path to **CHARLIE**, starting at the root, by beginning the path name with a slash (/). For example:

  ```
  CAT "/PROJECTS/Project_one/CHARLIE"
  ```

  This form assumes that you have already designated remote mass storage with some form of the MSI statement. If you have not, use the form:

  ```
  CAT "PROJECTS/Project_one/CHARLIE:REMOTE"
  ```
  for an SRM disc

  or

  ```
  CAT "PROJECTS/Project_one/CHARLIE:,700"
  ```
  for an HFS disc

  In the first example, the leading slash is not necessary, because including :**REMOTE** specifies the root as the beginning of the path. In the second example, the ":,700" is the volume specifier for the HFS disc.

- If you were in **MEMOS** (the directory immediately subordinate to **CHARLIE**), you could use the " .. " notation (explained with directory path syntax in the *BASIC Language Reference* manual). For example:

  ```
  CAT ".."
  ```

A Closer Look at Hierarchical Directory Capabilities

Directories are a type of file and, as such, can be:

- created with the CREATE DIR statement. When a directory is created, its location in the hierarchical structure is fixed.
- cataloged with the CAT statement, renamed with the RENAME statement, and protected with the PROTECT (SRM only) statement.
- "filled" with subordinate files and directories using the COPY, CREATE, CREATE BDAT, CREATE ASCII, CREATE DIR, SAVE, STORE, RENAME, RE-SAVE, and RE-STORE statements. Each subordinate file or directory is described in a fixed-format record in its superior directory.
- opened and closed with the MASS STORAGE IS (MSI) statement. When a user's MSI statement specifies a directory, any previously opened directory of that user is closed and the new one is opened.
- "emptied" by removing all subordinate files and directories with the PURGE statement.
- purged with the PURGE statement. You must close and empty a directory before purging it.

How SRM and HFS Directories and Files Are Stored

To most efficiently use disc space, the SRM system and HFS system store files non-contiguously and add space allocations to files as needed.

Non-Contiguous Storage of SRM and HFS Files

To avoid wasting disc space, SRM and HFS may "fragment" a file to fill unused disc sectors. This process is transparent and cannot be externally controlled. By "filling the gaps" automatically, the system eliminates the need to pack the shared disc's files.

Space Allocation for SRM and HFS Directories and Files

SRM and HFS files and directories grow dynamically as data is entered into them. This type of file is called *extensible*, because its size may be automatically extended (by BASIC) whenever it would otherwise overflow. (For SRM, the amount of space added to the file's current size is known as the "extent size" of the file; this amount of space is the same as the amount of space that was originally allocated to the file when it was initially created.)

Rather than restricting a file's space to that allocated when the file is created (for example, with a CREATE statement), the system determines disc space requirements when data is sent to the file (for example, by an OUTPUT statement). If additional data placed into a file would cause the file to overflow its current space allocation, the system automatically allocates more space for the file.

Similarly, directories grow only as entries are added. As a file or directory is created, another record is added to the containing directory.

Files are extended as long as there is sufficient unused disc space on the same volume. Excess data from a file will not be placed on any other volume, however.

A Closer Look at File Catalogs

There are three types of directory structures available with BASIC. (For further information about each of these directory formats, see the "Mass Storage Concepts" chapter.)

- LIF (Logical Interchange Format)—an HP corporate standard used by many HP operating systems; allows discs and files to be transportable between many different types of computer systems.
- HFS (Hierarchical File System)—also provides a format used by several operating systems, most notably the HP Series 200/300 HP-UX and Workstation Pascal systems. Like LIF, it also allows transporting discs and files between operating systems; however, it also provides a hierarchical directory structure, which LIF does not.
- SDF (Structured Directory Format)—also a hierarchical directory format; however, with Series 200/300 computers, it is available only with SRM "file server" systems.

BASIC displays slightly different results for each type of directory structure.

LIF Catalogs

Here is a typical catalog listing of a LIF directory (note for displays with 80 columns or more, there are two extra fields: DATE and TIME when the file was last modified):

```
:CS80,700
VOLUME LABEL: B9836
FILE NAME PRO TYPE  REC/FILE BYTE/REC   ADDRESS

MyProg        PROG        14      256        16
VisiComp      ASCII       29      256        30
GRAPH         BIN        171      256        59
GRAPHX        BIN        108      256       230
```

Here is what each portion of the catalog means:

`:CS80,700`	is the mass storage volume specifier (msvs) of the device.
`VOLUME LABEL`	is the "name" given to the volume (in this case, it is B9836).
`FILE NAME`	lists the names of the files in the directory (limit 10 characters).
`PRO`	indicates whether the file has a protect code (* is listed in this column if the file has a protect code).
`TYPE`	lists the type of each file.
`REC/FILE`	indicates the number of records (or sectors) in the file.
`BYTE/REC`	indicates the record size.
`ADDRESS`	indicates the number of the beginning sector in the file.
`DATE`	date when file was last modified (for displays with 80 or more columns)
`TIME`	time when file was last modified (for displays with 80 or more columns)

HFS Catalogs

Here is a typical catalog listing of an HFS directory:

```
:CS80, 700
LABEL: MyVol
FORMAT: HFS
AVAILABLE SPACE:     60168
                 FILE    NUM   REC     MODIFIED
FILE NAME        TYPE   RECS   LEN DATE        TIME PERMISSION OWNER GROUP
============== ===== ====== ===== ============== ========== ===== =====
lost+found       DIR       0    32 19-Nov-86 10:47  RWXRWXRWX    18     9
FILEIOD          PROG    191   256 21-Nov-86  9:03  RW-RW-RW-    18     9
RBDAT            BDAT      2   256 21-Nov-86  9:10  RW-RW-RW-    18     9
CATTOSTR         PROG      2   256  1-Dec-86  8:02  RW-RW-RW-    18     9
```

Here is what each column means:

`FILE NAME`	Lists the name of the file (limit 14 characters).
`FILE TYPE`	Lists the file's type (for instance, DIR specifies that the file is a directory; PROG specifies a BASIC program file; BDAT specifies a BASIC data file; etc.).
`NUM RECS`	number of *logical* records (the number of records allocated to the file).
`REC LEN`	the *logical* record size (default is 256 bytes for all files except HP-UX; BDAT files have user-selectable record lengths; record length for HP-UX files is 1).
`MODIFIED DATE TIME`	the day and time when the file was last modified.

`PERMISSION` specifies who has access rights to the file:

`R` indicates that the file can be read;
`W` indicates that the file can be written;
`X` indicates that the file can be searched (meaningful for hierarchical directories only).

There are 3 classes of user permissions for each file:

OWNER (left-most 3 characters);
GROUP (center 3 characters);
OTHER (right-most 3 characters).

See the subsequent section called "HFS File and Directory Permissions" for further information).

`OWNER` specifies the owner identifier for the file (for BASIC files, the default owner identifier is always 18).

`GROUP` specifies the group identifier of the file or directory (for BASIC, the default group identifier is always 9, which is used for "workstations" such as Series 200/300 BASIC and Pascal).

SRM Catalogs

Here is a typical catalog listing of an SRM directory:

```
PROJECTS/Project_one/CHARLIE:REMOTE 21, 0
LABEL:     Disc1
FORMAT:    SDF
AVAILABLE SPACE:     54096
                       SYS  FILE   NUMBER   RECORD      MODIFIED   PUB   OPEN
FILE NAME         LEV TYPE  TYPE  RECORDS   LENGTH DATE        TIME ACC   STAT
================= === ==== ===== ======== ======== ============== === ======
ASCII_1             1      ASCII        1      256  2-Dec-84 13:20
BDAT_1              1 98X6  BDAT         1      256  2-Dec-84 13:20 R
MEMOS               1       DIR         1       24  2-Dec-84 13:20 RW
```

Here is what the various columns mean:

FILE NAME — is the name of the file (up to 16 characters)

LEV — is always 1 (for BASIC).

SYS TYPE — indicates the type of system used to create the file. This is blank for ASCII files and directories. 98x6 denotes a Series 200/300 computer.

FILE TYPE — indicates the file type (such as PROG, ASCII, BDAT, etc.)

NUMBER RECORDS — is the number of records in the file.

RECORD LENGTH — is the record length used in the file (file size is the product of RECORDS×LENGTH).

MODIFIED DATE TIME — Date and time of day when the file was last written or modified.

PUB ACC — shows which access rights are currently public. For instance, MR would indicate that Manager and Read capabilities are public, while other rights are protected (and require a password to access them). See the subsequent section called "SRM Passwords" for details.

OPEN STAT — is the current OPEN/CLOSED/LOCKED status of the file.

- OPEN means that someone currently has the file open.
- LOCKED means that the file is opened in "EXCLUSIVE" mode, and no one else may access the file.
- if blank, no one is currently using the file.

Listing Only File Names (Requires MS Binary)

The following statement will produce a multi-column listing of the names of the files in the current working directory of the current default volume.

```
CAT; NAMES [Return]
```

```
lost+found      WORKSTATIONS     SYSTEM_BA5
MY_PROG         DATA_13          PROJECTS
```

Cataloging Selected Files (Requires MS Binary)

You can also specify which files to list with some of the following options to the CAT statement:

`CAT; SELECT "ABC"`	Lists only the files beginning with the specified letters "ABC".
`CAT; SKIP 30`	Skips the first 30 files in the directory and lists only the remaining files.
`CAT; COUNT Num_files`	Stores the number of lines in the catalog in the numeric variable called `Num_files`. (Note that this variable must be defined in the current program or subprogram context before it can be used.)
`CAT; NO HEADER`	Suppresses the catalog heading.
`CAT; NO HEADER, SKIP 10`	Suppresses the catalog heading and skips the first 10 files in the directory.

See the *BASIC Language Reference* for a complete description of these options.

Cataloging to a String Array (Requires MS Binary)

You can also send a catalog to a string array, as shown in the following example program segment:

```
      .
      .
      .
170  DIM Str_array$(1:40)[80]   ! 40 lines of 80 columns each.
180  CAT TO Str_array$(*)       ! Send catalog to string.
190   .
      .
      .
```

See the "Data Storage and Retrieval" chapter of *BASIC Programming Techniques* for further information on catalogs sent to string arrays.

Cataloging Individual PROG Files (Requires MS Binary)

Performing CAT operations on an individual PROG file returns additional information about the file. A catalog of a PROG files yields the following information:

- A list of the binary program(s) contained in the program file and the size of each (in bytes)
- The size of the main program (in bytes).
- A list of contexts (SUB and FN subprograms) and their sizes (in bytes)

The following catalog listing is an example of a CAT performed on an individual PROG file. Note that this catalog format only requires 45 columns.

```
NEWPAGER_A
NAME                     SIZE TYPE
======================== ===== ================
MAIN                    62002 BASIC
FNBar$                   3680 BASIC
FNRoman$                  656 BASIC
Killkeys                  426 BASIC
FNTrim$                   414 BASIC
FNUpc$                    344 BASIC
FNLwc$                    416 BASIC
Table_formatter          6810 BASIC
Strip                    1260 BASIC
  AVAILABLE ENTRIES = 0
```

The **AVAILABLE ENTRIES** table entry is not currently used.

The following listing shows a program which was stored while a BIN program was resident in the computer.

```
NEWPAGER_B
NAME                     SIZE TYPE
======================== ===== ================
PHYREC (2.0) Rev A       1734 BASIC BINARY
*** WARNING:  System level 5.  Bin level 1.
MAIN                    56394 BASIC
FNBar$                   3218 BASIC
FNRoman$                  656 BASIC
Killkeys                  426 BASIC
FNTrim$                   414 BASIC
FNUpc$                    344 BASIC
FNLwc$                    374 BASIC
Table_formatter          7622 BASIC
  AVAILABLE ENTRIES = 0
```

In addition, if the currently loaded BASIC system version is **different** from the binary program version, a warning and the version codes of both BASIC system and binary program are included in the catalog information. The following example shows the format of the message returned.

```
Prog_phy
NAME                     SIZE TYPE
======================== ===== ================
PHYREC 1.0               1734 BASIC BINARY
*** WARNING:  System level 5.  Bin level 1.
MAIN                      222 BASIC
  AVAILABLE ENTRIES = 0
```

General File Management Operations

This section describes the mechanics of managing files in your system. These may be program files, data files that your application creates, or data files that you create from the keyboard.

Closed vs. Open Files and Hierarchical Directories

Many of the following operations can only be performed on closed files and directories. Here is what the term "closed" means for files and directories:

- Files are open when there is an

  ```
  ASSIGN @Io_path TO "file_name"
  ```

  currently active for the file.

  Files are closed by this statement:

  ```
  ASSIGN @Io_path TO *
  ```

- Directories are closed when they are not the *current working directory.* A directory is the current working directory when you have made it the MASS STORAGE IS directory:

  ```
  MASS STORAGE IS "/USERS/MARK/MY_DIR"
  ```

The SCRATCH A command also closes any currently open directories and files. All files except those opened with the PRINTER IS statement are also closed by pressing [Reset] ([Shift]-[Break] on an ITF keyboard or [SHIFT]-[PAUSE] on a 98203 keyboard).

See the *BASIC Language Reference* description of these statements for details.

Protecting Files

You can "protect" files from being read, over-written, or destroyed by other users of the system. Since protecting files is slightly different for each of the three directory types, the discussion is split into three parts.

LIF Protect Codes

With LIF directories, protect codes are two-character strings that can be assigned to any BDAT, BIN, or PROG file with the PROTECT statement. The protect code, which does not appear in the CAT display, must be specified to subsequently modify the file. Protect codes are intended to prevent accidentally writing and purging files.

For example, to protect the file "SECRET" with the protect code "BS", use the statement:

```
PROTECT "SECRET","BS"
```

The protect code must subsequently be specified with the file name to allow access. For example, to RENAME the previously protected file "SECRET", the statement is:

```
RENAME "SECRET<BS>" TO "SHHHH"
```

File specifiers in mass storage statements that *write* to a file or directory must include the protect code, if the file has one. Mass storage statements that *read* a file or directory do not require the protect code (e.g., CAT, LOAD, LOAD BIN, LOADSUB, GET, and COPY).

To assign an I/O path name to the file named "SHHHH," you would now have to include the protect code.

```
ASSIGN @Path1 TO "SHHHH<BS>"
```

If you assign a protect code longer than two characters, the system will ignore everything after the second (non-blank) character. For example, the protect codes LONGPASS, LOST, and LOLLYGAG all result in the same protect code: LO. This rule holds both for PROTECTing a file and for specifying the protect code in a file specifier. For instance:

```
PROTECT "FILE1","Protect1"
```

would assign the protect code "Pr" to FILE1. To rename the file, we could write:

```
RENAME "FILE1<Prattle>" TO "FILE2"
```

"Prattle" is an acceptable protect code, since it starts with "Pr." Note that we do not include a protect code in the new file name. If you do, the system ignores it since the old protect code is passed to the new file name. FILE2 still has the protect code "Pr". To rename the file again, we might write:

```
RENAME "FILE2<Pr>" TO "FILE3"
```

Renaming a file has the effect of changing the file name in the directory and leaving everything else intact.

In addition to using the PROTECT statement, you can also assign a protect code to a BDAT file when you create it. For example:

```
CREATE BDAT "Example<xx>",10
```

The statement creates a 10-record file called "Example" and gives it a protect code of "xx". You can also do this to PROG files with the STORE statement. Since ASCII files cannot be protected, a protect code cannot be included in any CREATE ASCII, SAVE, or RE-SAVE statement.

To change a protect code, simply execute a new PROTECT statement. To change the protect code of "Example" to "yy," execute:

```
PROTECT "Example<xx>","yy"
```

Note that you must include the current protect code in the file specifier.

To completely remove a protect code from a file, PROTECT the file with a code consisting of two blanks. For example, to remove the protect code from file "Example," execute:

```
PROTECT "Example<yy>","  "
```

When specifying a file that does not have a protect code, you can either ignore the code entirely or include a code of two spaces:

```
PURGE "Example"
      or
PURGE "Example<  >"
```

HFS File and Directory Access Permissions

For HFS directories, you can use the PERMIT statement to assign and remove access permissions of a file or directory. Since this file system is compatible with the HP-UX system, BASIC uses a subset of the HP-UX file protection mechanism. (With HP-UX, the *chmod* command performs this function.)

There are 9 "permission bits" for HFS files and directories, broken into three classes (one for each class of users):

	OWNER			GROUP			OTHER	
READ	WRITE	SEARCH	READ	WRITE	SEARCH	READ	WRITE	SEARCH

The three **classes of users** are:

- OWNER—initially the person who created the file; however, ownership of individual files and directories can be changed with the CHOWN[1] statement. With BASIC, the system "owns" all files and directories with an owner identifier of 18.
- GROUP—initially the "group" to which the file's/directory's "owner" belongs; however, the group identifier of individual files and directories can be changed with the CHGRP[1] statement. With BASIC, the system is in the "group" with an identifier of 9, which is also the default group identifier used by the Workstation Pascal system.
- OTHER—all other users who are not the owner and are not in the same group as the owner—that is, everyone else.

[1] CHOWN and CHGRP are used *only* when you will also be using a disc with the HP-UX system. They give selected HP-UX users ownership or group access to files and directories. See the *BASIC Language Reference* entries for CHOWN and CHGRP for further information.

Each class of users has three **types of permissions** for accessing an HFS file or directory:

- READ—allows reading a file or directory (such as with CAT, ENTER, and GET).
- WRITE—allows a user to modify a file's or directory's contents (such as with OUTPUT, RE-STORE, or CREATE).
- SEARCH—an operation on directories which allows you to "search" the directory (such as with CAT and MASS STORAGE IS). This permission has no meaning for files (that are not directories) on BASIC.

The current state of these bits are shown in the `PERMISSION` column of a CAT listing of the directory in which the file or directory resides (`R` for READ; `W` for WRITE; `X` for SEARCH; - for "no permission"):

```
                FILE     NUM   REC      MODIFIED
FILE NAME       TYPE    RECS   LEN DATE         TIME PERMISSION OWNER GROUP
=============== ===== ====== ===== ================ ========== ===== =====
Directory         DIR     256     1 7-Nov-86    9:22 RWXRWXRWX     18     9
File             HPUX    8192     1 7-Nov-86    9:23 RW-RW-RW-     18     9
```

The default permission bits for directories are: `RWXRWXRWX`.
The default permission bits for files are: `RW-RW-RW-`.

The PERMIT statement is used to permit or restrict access of files and directories by other users on a system. For more information about user categories and how to change permissions on a file or directory, see the *BASIC Language Reference*: PERMIT statement.

The following example sets READ and WRITE permission for OWNER, but removes permission for SEARCH:

```
PERMIT "File"; OWNER:READ,WRITE
```

Before	**After**
---------	RW-------

With these permission bits set, the owner of the file can read and write the file (with GET and RE-STORE, for example), but all other users on the system cannot access the file.

The following example sets READ and WRITE permission for OWNER, but removes permission for SEARCH (note that the PERMIT parameters are the same as in the preceding example, but the "before" permission bits are different):

```
PERMIT "File"; OWNER:READ,WRITE
```

Before	After
R-XRW-RW-	RW-RW-RW-

With these permission bits set, all classes of users can read and write the file.

If a user class is not specified (OWNER, GROUP, or OTHER), the corresponding access-permission bits are not affected. For instance, the following statement sets the permission bits for OWNER and OTHER but leaves the bits for GROUP unchanged:

```
PERMIT "File"; OWNER:READ,WRITE; OTHER:READ
```

Before	After
R--R---W-	RW-R--R--

The next example changes bits for GROUP and OTHER but leaves the bits for OWNER unchanged:

```
PERMIT "File"; GROUP:READ; OTHER:READ
```

Before	After
RW-RW-RW-	RW-R--R--

If no user class is specified, the default permissions for all groups are restored:

```
PERMIT "File"
```

Before	After
RW-R--R--	RW-RW-RW-

```
PERMIT "Directory"
```

Before	After
RW-R--R--	RWXRWXRWX

SRM Passwords and Locks

The SRM system offers three kinds of access capability for files and directories:

READ — For a file, possessing this access capability allows you to execute statements that read the file (such as GET, ASSIGN, ENTER, etc.).

For a directory, possessing this access capability allows you to execute statements that read the file names in the directory, and to "pass through" the directory when the directory's name is included in a directory path. For example, in the SRM file specifier:

```
"/PROJECTS/Project_one<READpass>/JOHN/f1"
```

including the assigned password **<READpass>** allows passage through the directory **Project_one** to allow access to its subordinate directories and files.

WRITE — For a file, possessing this access capability permits you to execute statements that write to the file (such as SAVE, OUTPUT, etc.).

For a directory, possessing this access capability allows you to execute statements that add to or delete from the directory's contents (such as CREATE ASCII, CREATE DIR, PURGE, etc.).

MANAGER — With the MANAGER access capability, public capabilities for a file or directory differ slightly from password-protected capabilities.

- Public MANAGER capability allows any SRM user to PROTECT, PURGE, or RENAME the file.
- The password-protected MANAGER capability provides MANAGER, READ, and WRITE access capabilities to users who include a valid password in the file or directory specifier.

Capabilities are either public (available to all workstations on the SRM) or protected (available only to users who know the appropriate password).

The current access capabilities for a file are shown in a catalog listing:

```
PROJECTS/Project_one:REMOTE 21, 0
LABEL:    Disc1
FORMAT:   SDF
AVAILABLE SPACE:      4354096
                       SYS  FILE   NUMBER   RECORD     MODIFIED      PUB   OPEN
FILE NAME         LEV TYPE  TYPE  RECORDS   LENGTH DATE        TIME  ACC   STAT
================= === ==== ===== ======== ======== ================ ==== ======
ASCII_1             1      ASCII        0      256  2-Dec-84 13:20
BDAT_1              1 98X6  BDAT         0      256  2-Dec-84 13:20 R
MEMOS               1       DIR          0       24  2-Dec-84 13:20 RW
```

In the above example, the file `ASCII_1` has no public access capabilities; that is, all access must include the appropriate password. The file `BDAT_1` has the READ capability public, which means that anyone on the SRM system can read the file. The directory `MEMOS` has READ and WRITE capabilities still open to the public; anyone can create and purge files in the directory, as well as "search" through the directory (with statements like `MASS STORAGE IS "MEMOS/SUB_DIR"`).

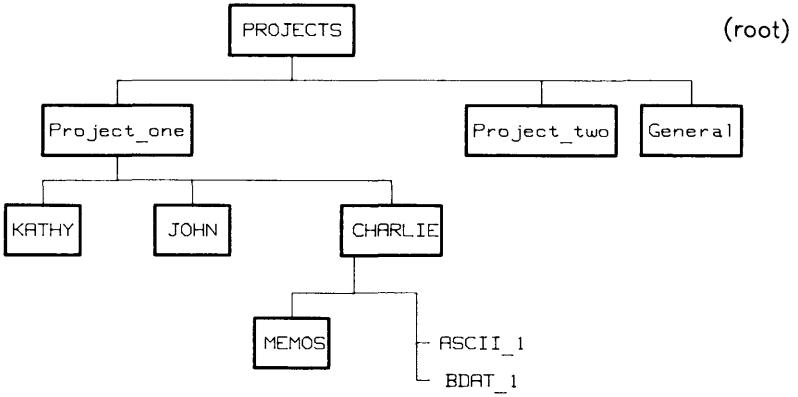
Capabilities are protected with the PROTECT statement, which associates a password with one or more access capabilities. Each file or directory can have several password/capability pairs assigned to it.

Once assigned, the password protecting an access capability must be included with the file or directory specifier to execute statements requiring that access. If you don't specify the correct password when it is required, the system will report an error and deny access to the file or directory.

When you create directories and files, their access capabilities are "public" (available to any user on the SRM). You may subsequently protect a directory or file against certain types of access by other SRM workstations, provided *all* of the following are true:

- You possess MANAGER access capability on the file or directory (MANAGER access to the file is public or you know the password protecting the capability).
- You possess READ access capability on the directory immediately superior to the file or directory you wish to protect.
- You protect the file or directory either while "in" its superior directory or by specifying the valid directory path to its superior directory.

For example, using the directory structure established for other examples in this section (see illustration) and assuming no passwords have been assigned to the files, you could:



1. Assign the password **passme** to protect the MANAGER and WRITE access capabilities on the directory **CHARLIE** with the sequence:

```
MSI "/PROJECTS/Project_one"

PROTECT "CHARLIE",("passme":MANAGER,WRITE)
```

   which executes the PROTECT statement after moving to the directory **Project_one** (immediately superior to **CHARLIE**). As a result of this PROTECT statement, the READ access capability on **CHARLIE** is still public, but any operations that require MANAGER or WRITE capabilities must include the password.

2. Remove all public access capabilities from the file ASCII_1 by assigning the password no_pub, using:

```
PROTECT "CHARLIE/ASCII_1",("no_pub":MANAGER,WRITE,READ)
```

or

```
MSI "CHARLIE"
PROTECT "ASCII_1",("no_pub":MANAGER,WRITE,READ)
```

These statements assume you are in the directory Project_one. The second sequence of statements makes CHARLIE the new working directory, whereas in the first, you merely "pass through" CHARLIE to reach ASCII_1. With the READ access capability on CHARLIE still public, you do not need a password.

3. Protect the file BDAT_1 so data can be read from it but not written into it without using the password write. If the current working directory were CHARLIE, you would type:

```
PROTECT "BDAT_1",("write":MANAGER,WRITE)
```

4. Protect the MANAGER access capability of the directory MEMOS with the password, mgr_pass (so that everyone can read from and write to the directory, but a password is required to purge the directory or its contents) by typing:

```
PROTECT "MEMOS",("mgr_pass":MANAGER)
```

If you protected the files and directory in CHARLIE as in the steps above, a catalog listing of CHARLIE would look something like this:

```
PROJECTS/Project_one/CHARLIE:REMOTE 21, 0
LABEL:     Disc1
FORMAT:    SDF
AVAILABLE SPACE:     54096

                       SYS  FILE    NUMBER    RECORD      MODIFIED      PUB    OPEN
FILE NAME         LEV TYPE  TYPE   RECORDS    LENGTH DATE       TIME   ACC    STAT
================= === ==== ===== ======== ======== ============== ==== ======
ASCII_1             1       ASCII         6      256  2-Dec-84 13:20
BDAT_1              1 98X6  BDAT          4      256  2-Dec-84 13:20 R
MEMOS               1       DIR           0       24  2-Dec-84 13:20 RW
```

The letters in the column labeled PUB ACC indicate access capabilities that are public (not protected with a password). For example, only the MANAGER (M) access capability on the directory MEMOS has been protected, leaving the READ (R) and WRITE (W) capabilities available to any SRM workstation user.

Specifying Passwords

When a password is required, you must include the correct password as part of the file or directory specifier in any command or statement that requires the protected access on the file or directory. The password must be enclosed between "<" and ">" and must immediately follow the name of the file or directory it protects.

For example, to get the file `ASCII_1`, you might execute:

```
GET "/PROJECTS/Project_one/CHARLIE/ASCII_1<no_pub>"
```

If the password were not included in the specifier, the system would respond with an error message and refuse to get the file.

Exclusive Access: Locking SRM Files

Although sharing SRM files saves disc space, allowing several users access to one copy of a file introduces the danger of users trying to access the file at the same time, which can cause unpredictable results. For instance, if one user tries to read part of a file while another user is writing to it, the file's contents may be inaccurate for the read.

To avoid problems, the SRM system adds two BASIC keywords, LOCK and UNLOCK, which you can use to secure files during critical operations. LOCK establishes exclusive access to a file, which means that the file can only be accessed from the workstation at which the LOCK was executed. You may wish to LOCK a file, for example, during any procedure that writes new information to the file. The typical procedure is to LOCK all critical files, read data from files, update the date, write the data into the files, and then UNLOCK all critical files.

To permit shared access to the file once again, UNLOCK must be executed from the same workstation, or the file must be closed. Only ASCII or BDAT files that have been opened by a user via ASSIGN may be locked explicitly by that user.

Locking and unlocking is usually done from within a program. For more information, refer to the descriptions of the ASSIGN, LOCK and UNLOCK keywords in the *BASIC Language Reference.*

Locking and Unlocking SRM Files

You can "lock" an SRM file with the LOCK statement, giving you sole access to that file. The same file can be locked several times in succession. Unlocking a file requires that you cancel all locks on that file. If you use the UNLOCK statement, you must cancel each LOCK with a corresponding UNLOCK. Using ASSIGN to re-open a locked file unlocks the file and you must execute another LOCK statement to lock the file again. Closing the file via ASSIGN @...TO * cancels all locks on the file.

In this example, a critical operation must be performed on the file named `File_a`, and you do not want other users accessing the file during that operation. The program might be as follows:

```
1000   ASSIGN @File TO "File_a:REMOTE"
1010   LOCK @File;CONDITIONAL Result_code
1020   IF Result_code THEN GOTO 1010  ! Try again
1030   !  Begin critical process
       .
       .
       .
2000   !  End critical process
2010   UNLOCK @File
```

The numeric variable called `Result_code` is used to determine the result of the LOCK operation. If the LOCK operation is successful, the variable contains 0. If the LOCK is not successful, the variable contains the numeric error code generated by attempting to lock the file.

Copying Files and Volumes

The COPY statement allows you to duplicate an individual file or an entire disc volume. Any type of file may be copied.

- **Copying a file** duplicates the existing file and places the new file name in the directory.

  ```
  COPY "ExistFile" TO "NewFile"
  ```

  A new file can be created either on the same volume or on another volume. If you copy a file to the same volume, the new file name must be different from the existing file name if it is in the same directory. If there is not enough room on the disc for the file to be copied, the system cancels the statement and reports an error.

- **Copying an entire volume** makes an *image* copy of the source volume on the destination volume. This type of copy is allowed with LIF and HFS volumes.

Note: This type of copy **destroys all** data on the destination disc. If you want to copy multiple files, you should use one of the "back-up" utilities described in the "BASIC Utilities" chapter of *Installing and Maintaining the BASIC System.*

To perform this type of copy, simply identify the source and destination volumes.

```
COPY ":,700" TO ":,702"
```

Note: You can copy a larger volume to a smaller volume, if copying the files from the larger volume will not overflow the smaller one. However with HFS, copying a smaller volume to a larger volume will effectively change the size of the larger volume (making it the size of the smaller volume).

More Examples

The following statement copies "File1" from the current default mass storage volume to a new file called "File2" on the same volume:

```
COPY "File1" TO "File2"
```

The following statement copies "File1" from the current default mass storage to a drive at interface select code 7, primary address 0, unit number 1. Note that both files can be named "File1" if they are on *different* volumes.

```
COPY "File1" TO "File1:,700,1"
```

COPY can be used to copy files from volumes of one format to another. For instance, you can copy LIF files to HFS or SRM volumes, SRM files to HFS or LIF volumes, and so forth.

Copying LIF Files

The following statement copies a file from a CS80 drive to the current default mass storage device.

```
COPY "File1:,700,0" TO "DATA"
```

The following statement copies the entire disc from the right-hand drive to the left-hand drive of a dual-drive disc.

```
COPY ":,700,1" TO ":,700,0"
```

You can copy an entire LIF volume in a single COPY statement. Make **absolutely sure** that you understand the consequences of this type of operation (destroys all previous data on the destination).

Copying HFS Files

To copy an HFS file, you must have R (read) permission on the existing file and X (search) permission on all superior directories. You also need W (write) and X permissions on the destination directory into which the duplicate file is being copied, as well as X permission on all superior directories.

You cannot copy directories, although you can copy files from one directory to another. You can copy an entire HFS volume in a single COPY statement (as with LIF, this operation destroys all previous data on the destination).

Copying SRM Files

To copy an SRM file, you must have R (read) permission on the existing file. You must also have W (write) permission on the directory into which the duplicate file is being copied, as well as R permission on all superior directories.

You cannot copy directories, although you can copy files from one directory to another. You cannot copy an entire SRM volume in a single COPY statement. (You must copy an SRM volume file by file.)

Renaming Files

The name of a file can be changed without disturbing the file's contents. This is done with the RENAME statement. For example, to change the name of a file (on the default volume) from "George" to "Frank", use the following statement:

```
RENAME "George" TO "Frank"
```

Special Additional Action with SRM and HFS Volumes

The RENAME statement can be used to change the name of a file, optionally changing its location in the hierarchy:

```
RENAME "/USERS/MARK/File22" TO "/USERS/MARK/OLD_FILES/File22"
```

Purging Files

You can purge a file from a directory by using the PURGE statement. Purging a file deletes the directory entry for the file and releases the space reserved for the data area.

A file entry can be removed from the disc directory with the PURGE statement. This prevents any further access to the file. For example, the following statement removes the file "Old_stuff" from the current default volume:

```
PURGE "Old_stuff"
```

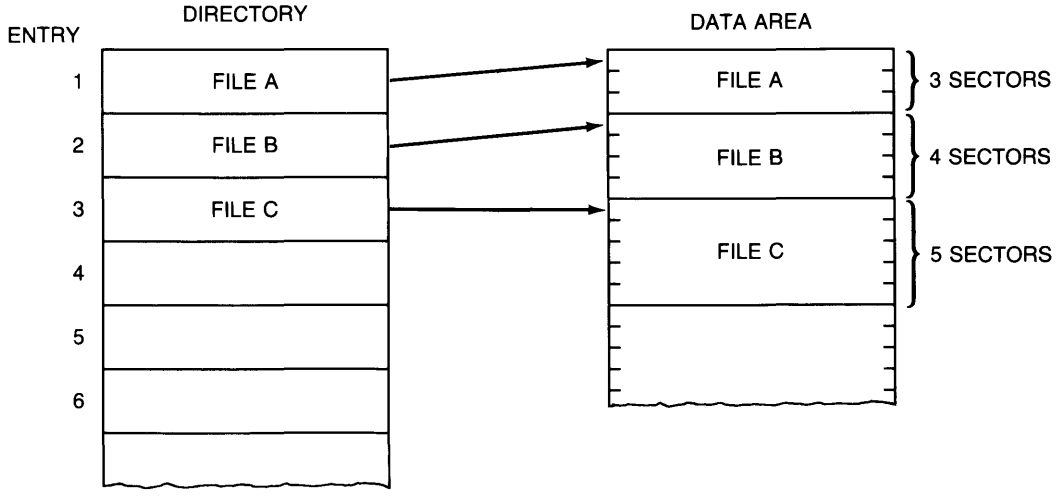
Once a file is purged, there is generally no way of retrieving the information it contains[1].

Effects of PURGE on LIF Directories

The order of file names and files' data areas is the same on LIF discs. Therefore, purging a file on a LIF directory creates two "gaps" on the disc:

- One in the data area
- One in the directory.

As an example, suppose that you have three consecutive files on a disc with the following names and sizes.



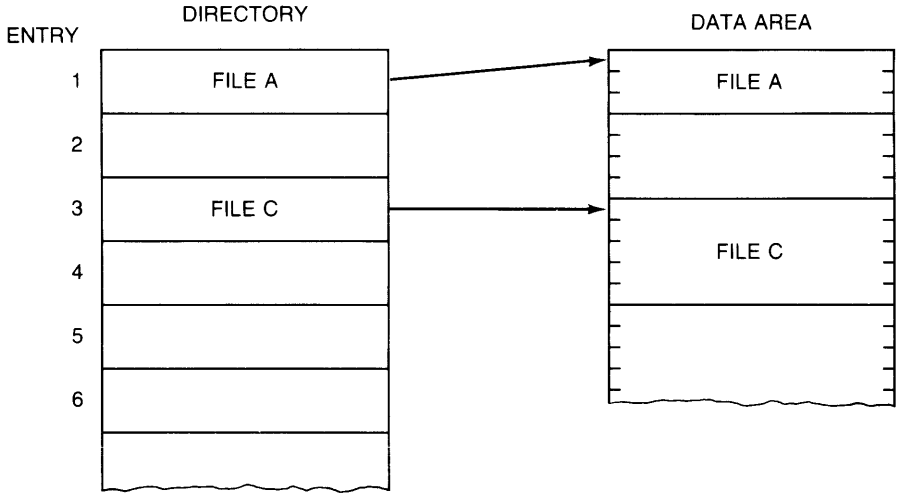
[1] The only exception to this rule is with LIF discs, on which files can be "un-purged" using the Mass Storage Program (MASS_STOR). See the "Utilities" section of *Installing and Maintaining the BASIC System* for instructions.

LIF directory entries are in the same order as the files in the data area. The third directory entry, for example, must correspond to the third file in the data area. Consequently, if you PURGE a LIF file and then create a smaller file, you may lose disc space. The following examples illustrate this principle.

Executing the following statement:

```
PURGE "FileB"
```

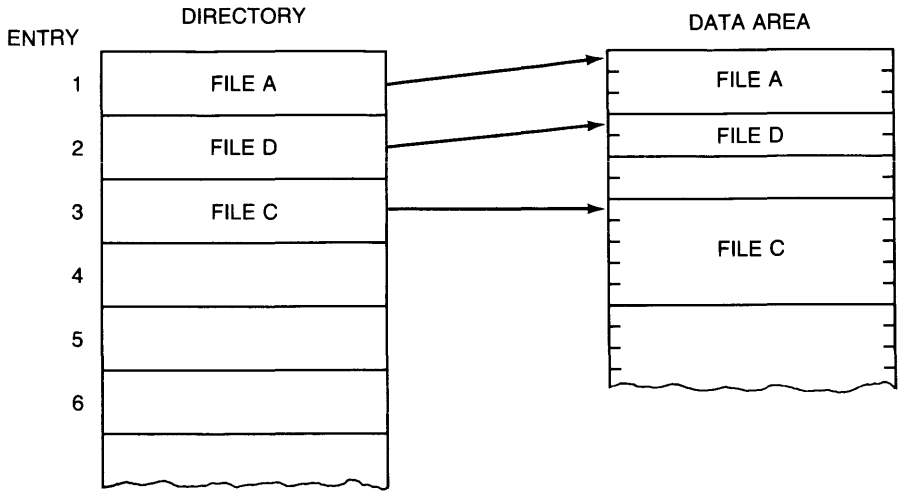
creates a 1-entry gap in the directory and a 4-sector gap in the data area.



When you create a file on a LIF volume, the system looks for the first gap in the data area with enough room to store the file. When it finds one, it puts the file into this gap. To continue the above example, suppose you create a 2-sector file with this statement:

```
CREATE ASCII "FileD",2
```

The system will place this file in the data-area gap and place the directory entry in the directory gap.



You now have a 2-sector gap in the data area but no gaps in the directory. If you create another file, the system will fill entry 4 in the directory and will reserve space in the data area past `FileC`. The two unused sectors will not be reclaimed unless you PURGE one of the adjacent files, `FileD` or `FileC`.

In this example, the 2 unused sectors were not a large problem. However, suppose that `FileB` was 800 sectors long, and `FileD` was 2 sectors. This scenario would result in 798 sectors on the disc being unusable!

The solution to this problem is to "REPACK" the disc using the "Mass Storage" (MASS_STOR) utility, available on the *BASIC Utilities* disc. See the "Utilities" section of *Installing and Maintaining the BASIC System* for instructions.

Purging HFS Files and Directories

The PURGE statement can be used for removing HFS files and directories.
Here are the restrictions on using PURGE to remove HFS files:

- In order to use PURGE, you must have write permission on the parent (superior) directory.
- PURGE works only with closed files and directories. (You cannot PURGE a file currently open with ASSIGN or a directory which is the current working directory—specified in the most recent MASS STORAGE IS statement.)
- Directories must also be empty (not contain any files or directories).

Purging SRM Files and Directories

The PURGE statement can be used for removing SRM files and directories. PURGE works only with closed files and directories. Directories must also be empty (not contain any files or directories).

When specifying the SRM file to be purged, you must include a write password for the file and read/write passwords for its parent directory. For example, to purge the file **BDAT_1** from the directory **CHARLIE** (see previous examples), you could type:

```
PURGE ".<passme>/BDAT_1<write>"
```

In this example, **CHARLIE** is the current working directory, as denoted in the directory path by " . ". (Refer to the syntax for directory path in the *BASIC Language Reference* manual).

To purge a file, you must have the WRITE access capability on that file and READ and WRITE access capabilities on the file's superior directory. Because **passme** protects the WRITE capability on **CHARLIE** and **write** protects the WRITE capability on **BDAT_1**, both passwords must be included in the file specifier in the PURGE statement.

Although you do not normally need to specify the working directory in a directory path, you must include the password for the PURGE operation. The READ capability on **CHARLIE** is not password-protected.

To purge **CHARLIE**, you would first need to purge the remaining files and directory in **CHARLIE**. Because the MSI statement "opens" a directory (making it the current working directory), you must also "close" **CHARLIE**.

For example, if no files or directories remained in **CHARLIE**, you could purge **CHARLIE** by executing these two commands:

```
MSI ":REMOTE"
PURGE "PROJECTS/Project_one/CHARLIE<passme>
```

The first statement closes **CHARLIE** and establishes the root directory as the current working directory. Note that, because **passme** protects the WRITE access capability on **CHARLIE**, you must include that password in the PURGE statement.

Volume Labels (LIF and HFS Volumes Only)

When you INITIALIZE a LIF disc, the BASIC system gives it a default "volume label"—"B9836" or something similar—that is displayed with the CAT statement:

```
        :CS80,700
----->  VOLUME LABEL: B9836
        FILE NAME PRO TYPE  REC/FILE BYTE/REC   ADDRESS

        MyProg        PROG        14      256        16
        VisiComp      ASCII       29      256        30
```

Volume labels are useful in cases when you want to identify a particular *disc media*, rather than a particular *disc drive*, for instance. (The example below shows this use.)

Reading Volume Labels

To determine the label on the media currently installed in a mass storage device, use the device's mass storage unit specifier:

```
10  READ LABEL Label$ FROM ":,700,1"
```

The statement puts the label into the string variable named Label$, which must be large enough to hold all of the characters in the label (volume labels are 6 characters or less).

Here is an example that searches two drives for a disc with the volume label "MY_VOL":

```
      .
      .
      .
100   READ LABEL Vol_label$ FROM ":,700" !       Unit 0.
110   IF Vol_label$="MY_VOL" THEN Msus$=":,700"
120   !
130   READ LABEL Vol_label$ FROM ":,700,1" !     Unit 1.
140   IF Vol_label$="MY_VOL" THEN Msus$=":,700,1"
150   !
      .
      .
      .
```

Writing Volume Labels

To write the label "MY_VOL" onto the current default volume, you could execute:

```
PRINT LABEL "MY_VOL"
```

To change the volume label on a different volume, you will need to specify the volume:

```
PRINT LABEL "MY_VOL" TO ":,700,1"
```

Enabling Checkread Verification

Normally, the BASIC system writes data into files and does not verify that the data was written without error. The reliability of mass storage devices is generally high enough to justify this. However, if you want the system to perform a read-after-write verification of data written into files, you can use this statement:

```
CHECKREAD ON
```

Data subsequently written into all files by the following statements is subject to this verification:

COPY PRINT LABEL RE-SAVE CREATE ASCII PROTECT STORE CREATE BDAT PURGE RE-STORE OUTPUT RENAME TRANSFER SAVE

Note, however, that CHECKREAD does not affect files being written through the PRINTER IS or PLOTTER IS statements. For SRM systems, there is already a checkread operation performed automatically by the SRM controller; therefore, using the CHECKREAD statement has no effect with SRM.

To disable this feature, execute:

```
CHECKREAD OFF
```

What to Do Next

Task/Topic	Chapter/Section
Learn how to load and run programs.	"Loading and Running Programs"
Learn how to enter, edit, secure, document, and store programs.	"Editing and Storing Programs"
Learn how to maintain your system.	*Maintaining BASIC* section, *Installing and Maintaining the BASIC System*
Learn how to use a BASIC utility program.	"BASIC Utilities Library"
Learn about each key on your keyboard.	"Keyboard Reference"

Notes

Task/Topic	Where to Go for Details
Entering and storing programs (overview)	6-1
Terminology	6-1
Getting into EDIT mode	6-3
Correcting typing mistakes	6-4
Storing the line	6-4
Entering program lines	6-4
Upper-case or lower-case letters?	6-6
Keys used for editing the current line	6-7
Keys used for scrolling the program	6-9
Inserting lines	6-10
Deleting and recalling lines	6-11
Copying lines (by changing line numbers)	6-12
Getting out of EDIT mode	6-12
Listing the program	6-13
Storing the program	6-13
Running the program	6-13
A closer look at editing	6-14
Typing-aid softkey menu changes (ITF keyboards)	6-15
A closer look at listing a program	6-16
Global editing operations	6-18
Indenting a program	6-19
Finding textual patterns	6-22
Search-and-replace operations	6-24
Copying program segments	6-25
Moving program segments	6-25
Deleting multiple lines	6-26
Making programs readable	6-27
Software security	6-31
A closer look at storing programs	6-33
Using STORE	6-34
Using SAVE	6-34
What to do next?	6-36

Editing and Storing Programs 6

In this chapter you will learn how to enter, edit, and store BASIC programs.

Entering and Storing Programs (Overview)

One of the great joys of using this BASIC system is that it makes entering, editing, and storing programs an extremely simple task. This section introduces you to some fundamental concepts and skills involved in entering and storing BASIC programs.

Terminology

keyword — a group of characters recognized by BASIC to represent some predefined action. Examples are:

```
CAT
LOAD
COPY
```

statement — a keyword followed by any parameters and/or secondary keywords that:

- are required or allowed with that keyword
- and fit on one program line[1]

Examples are:

```
CAT ":,700"
LOAD "MyProg"
COPY "MyProg" TO "BackupFile"
```

command — a statement that is executed from the keyboard.

[1] The maximum length of a command or program line is two CRT lines (up to 256 characters. When entering commands and programs from the keyboard, this is 160 characters on most models, but only 100 characters on the Model 226. (Program lines longer than 100 characters can be created on a 310, for instance, and then transferred to a 226 using a mass storage operation such as LOAD or GET. The resulting lines are valid program lines on the 226. However, when viewed on the CRT in EDIT mode, these lines have an asterisk after the line number, with only the first 100 characters visible.)

program line a statement preceded by a line number (and optional line label) that is stored in a program. Examples are:

```
100  CAT ":,700"
250  COPY "MyProg" TO "BackupFile"
875 Line_label:  LOAD "MyProg"
```

Statements as Commands vs. Program Lines

In general, a statement can be used as *either*:

- A command—if the statement is executed from the keyboard; for example:

  ```
  PRINT "This is a keyboard command." Return
  ```

- A program line—if the statement contains a leading line number; for instance:

  ```
  100  PRINT "This is a program line."
  ```

However, there are some statements that *cannot* be:

- Executed as commands (such as DIM and RETURN)

  ```
  100  DIM String_var$[100]
  ```

- Stored as program lines (such as DEL and SCRATCH)

  ```
  SCRATCH A
  ```

The general case is that statements are both programmable and keyboard executable, such as CALL and PRINT. The *BASIC Language Reference* shows whether or not each keyword is keyboard executable or programmable, or both.

Getting into EDIT Mode

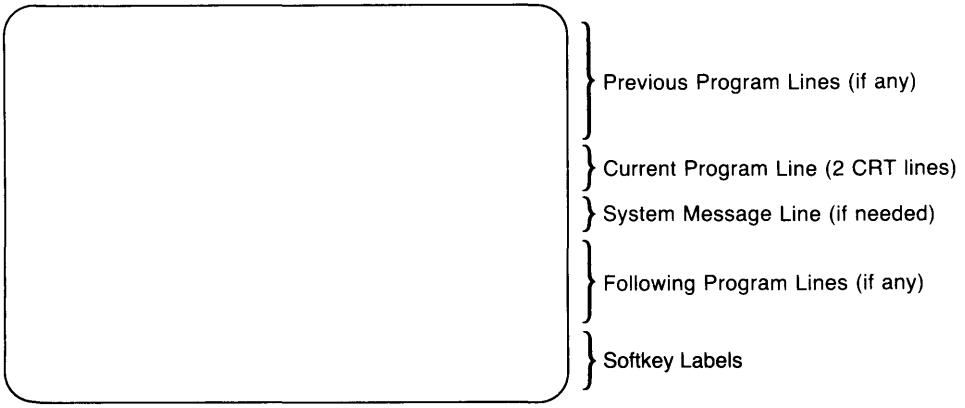
When you want to edit program lines, you will need to get into EDIT mode[1]. If a program is not running[2], you can get into edit mode by typing:

```
EDIT [Return]
```

or by pressing the [EDIT] key followed by [ENTER] (HP 98203 keyboards only).

You can also use the **EDIT** softkey ([f1] in the `User 1` menu of an ITF keyboard).

The system goes into EDIT mode, and the screen has this format:



[1] The EDIT binary must be loaded in order to use EDIT mode. It is possible, however, to enter a program by typing program lines (line number and statement) on the normal keyboard input line of the CRT and pressing the [Return] key. But it is usually much more desirable to use EDIT mode where you can see several program lines at one time.

[2] See the "Introduction to the System" chapter for instructions on determining whether or not a program is running.

In this mode, you can view a multi-line portion of the program. You can view different portions of the program by scrolling the display (see subsequent section called "Scrolling the Program" for details). If you want to edit a particular line, you must scroll the display so the line you want to edit is in the middle of the screen.

If there is no program in memory when you enter EDIT mode, the cursor will appear on a line with the number 10, which is the default line number of the first program line.

You can then begin to enter program lines; the following section explains how to do that.

Note

When you enter EDIT mode, the typing-aid softkey definitions and labels are changed to a User menu, whose definitions are more useful for editing operations (need KBD binary). See the subsequent section called "A Closer Look at Editing" for further information.

Correcting Typing Mistakes

If you make any errors while typing, use the [Back space] or [◄] and [►] keys to move the cursor to the erroneous character(s), and then re-type them correctly.

Storing the Line

Once the line is exactly as you want it, press [Return]. (The cursor may be any place on the line when you store it; the system will read the entire line, regardless of the location of the cursor.)

Entering Program Lines

To enter a program line, type the desired characters at the keyboard. For practice, type in the lines shown below (the line numbers to the left are supplied for you).

```
10 PRINT "Tiny prog." [Return]
20 END [Return]
30
```

The BASIC System Checks Syntax

Before storing a program line, the computer checks for syntax[1] errors, and also changes the letter-case of keywords and identifiers (see the following section, "upper-case or lower-case Letters?" for details).

Immediate syntax checking is one big advantage of writing programs on this BASIC system. A great many programming errors can be detected at program-entry time, which increases the chances of having a program run properly and cuts down debugging time. If the syntax of the line is correct, the line is stored, and the next line number appears in front of the cursor.

If the system detects an error in the input line, it displays an error message immediately below the line and places the cursor at the location it blames for the error.

```
   10  PRINT "Short program.
              -
Error 985  Invalid quoted string

   20  END
```

Keep in mind that there is an endless variety of human mistakes that might occur, and that BASIC is not very good at dealing with even slight ambiguities. As a result, you might not always agree with its diagnosis of the exact error or the error's location. However, an error message definitely indicates that something needs to be fixed. There is a complete list of error messages and their meanings in the "Errors" appendix of the *BASIC Language Reference* and *BASIC Condensed Reference* manuals.

[1] *Syntax* is a term used to describe the way in which keywords, parameters, etc. are put together to form a legal statement.

Upper-case or Lower-case Letters?

Program entry is simplified by the computer's ability to recognize the upper-and lower-case letter requirements for most elements in a statement. An entire statement can be typed using all upper-case or all lower-case letters. If the statement's syntax is correct, and there are no "keyword conflicts" (see the explanation below), the system stores the program line. Upon LISTing or EDITing the program[1], however, the system uses these conventions:

- Keywords are *all* upper-case letters (CAT, LOAD, DISP, etc.).
- All variable names are listed with the first letter in upper-case and the rest of the letters, if any, in lower-case[2] (Var1, String33$, etc.).

In other words, you don't usually have to bother with the [Shift] key when you enter a line, because the system will automatically change all letters to the proper letter-case. On the other hand, if there is a "keyword conflict," an error is reported. A keyword conflict occurs when you try to use a keyword for an identifier (variable name, line label, or subprogram name). You can use keywords for identifiers; simply change the letter-case of at least one letter in the identifier name (for example, **Cat** or **cAt**), and then press [Return] again. A word containing a mixture of upper-case and lower-case letters is assumed to be an identifier.

The system's assumptions about keywords versus identifiers won't cause any problems if your line has the proper syntax. However, if you are guessing at a keyword or syntax, don't assume that you got the line right just because the computer stored it. For instance, suppose that you are trying to type a PRINT statement to print a blank line; however, you misspell the keyword PRINT:

```
100  PRINY
```

The system does not report an error, because the line could legitimately be interpreted as a call to a subprogram named "Priny".

Thus, in general, if the system puts lower-case letters in something you thought was a keyword, then it wasn't really recognized as a keyword.

---

[1] The EDIT binary must be loaded in order to edit and list programs.

[2] Accented characters (in the range CHR$(16) to CHR$(244)) remain as entered. These characters do not occur in keywords.

Keys Used for Editing the Current Line

In order to edit a line, it must be the "current line"—the line that the cursor is on. The next few paragraphs give a quick overview of the standard editing keys that you can use while editing the current line. (The "Keyboard Reference" chapter lists all key definitions for each type of keyboard.)

Editing Feature	Explanation
Normal cursor (blinking underscore _)	Whenever you type characters at the keyboard, they appear on the current line at the cursor, *overwriting* any existing characters.
[◄], [►], and [Back space]	move the cursor one character in the indicated direction. If the cursor has reached either end of the line, it doesn't go any farther. Pressing [Shift] [►] moves the cursor to the end of the line, and [Shift] [◄] moves the cursor to the beginning of the line.
Knob or mouse	also move the cursor.
[INS CHR] or [Insert char]	changes the cursor to the insert cursor (see below), and enters insert mode (any characters typed are placed before the current cursor position, and the cursor and subsequent characters are shifted one position to the right). This key toggles between the normal cursor and the insert cursor.
Insert cursor (inverse-video block, ▒)	indicates that the character entered is inserted *in front of* the character currently highlighted by the cursor.
[DEL CHR] or [Delete char]	deletes the character pointed to by the cursor. Subsequent characters on the line are shifted one position to the left.
[CLR→END] or [Clear line]	deletes all the characters from the cursor to the end of the current line.
[CLR LN] or [Shift]-[Clear line]	clears the entire current line.

Editing Feature	Explanation
[SET TAB] and [CLR TAB] or [f5] and [Shift]-[f5] (System menu)	[SET TAB] and [CLR TAB] perform the indicated action at the cursor position.
[Tab] and [Shift]-[Tab]	The [Tab] key moves the cursor to the next tab position, if there is one. [Shift]-[Tab] moves the cursor back to the previous tab position, if there is one.
[ANY CHAR] ([SHIFT]-[STEP]) or [f7] (System menu)	Characters that don't appear on the keycaps can be typed by using this key. Assume you are typing a program line and you want the vertical bar character (and it is not on your keyboard, for example). Press the [ANY CHAR] key. The following message appears below the current line: `ENTER 3 DIGITS, 000 THRU 255` For instance, the decimal ASCII code for a vertical bar is 124. Press the [1] [2] [4] number keys. A vertical bar appears at the cursor position, and the message goes away. If a key that is not part of a 3-digit number in the proper range is pressed during this operation, the ANY CHAR operation is aborted and the key performs its normal function. (By the way, the vertical bar character is available on the 98203 keyboard; press [SHIFT]-[(] on the numeric keypad.)
Softkeys [k0] thru [k9] or [f1] thru [f8] (in the User 1, 2, and 3 menus of ITF keyboards)	These keys produce characters and system-key presses, just as if you had typed them at the keyboard. See the "Using Typing-Aid Softkeys" section of the "Introduction to the BASIC System" chapter for details.

Keys Used for Scrolling the Program

All of EDIT mode's text-entry capabilities apply to the "current line"—the line in the middle of the screen with the cursor on it. That is, you must move a line to the current-line position before you can edit it[1]. The text on the screen is scrolled so that you are always editing the line in a "window" in the middle of the screen.

Editing Key	Explanation
[▲]	scrolls the program up one line, so that you will be editing the next program line. (Note that the cursor remains on the line in the middle of the screen.)
[▼]	scrolls the program down one line, so that you will be editing the preceding program line.
[Shift]-[▲]	scrolls the program all the way up, so that the end of the program is displayed and the next available line number is shown on the current line.
[Shift]-[▼]	scrolls the program all the way down, so that the beginning of the program is displayed and the first program line is the current line.
Knob or mouse	scroll the program.

[1] The only exception to this is when you enter a new line with the same line number as an existing line. In that case, the new line replaces the old, even though the old line was not moved to the current-line position.

Inserting Lines

Lines can be easily inserted into a program. As an example, assume that you want to insert some lines between line 90 and line 100 in your program. Place line 100 in the current-line position, and press the [Insert line] ([INS LN]) key.

```
90   PRINT "Line 90."
100  PRINT "Line 100."  <------ Make this the current line,
                                then press [Insert line].
```

The program display "opens" and a new line number appears between line 90 and line 100.

```
90   PRINT "Line 90."
91   _                  <------ Begin typing; letters appear at cursor.
100  PRINT "Line 100."
```

Type and store the inserted lines in the normal manner. Appropriate line numbers will appear automatically.

The insert mode can be canceled by pressing the [Insert line] ([INS LN]) key again, or by performing an operation that causes a new current line to appear (such as scrolling).

While inserting lines, the system maintains the established interval between line numbers, if possible.

- If the interval between lines in the preceding example was 5, the first line number appearing would be 95.
- When the normal interval between lines can no longer be maintained, an interval of 1 is used. Thus, after line 95 is stored, the next line number supplied is 96.
- When there are no line numbers available between the current line and the next line, enough of the program below the current line is renumbered to allow the insert operation to continue. In the example, this would happen after line 99 is stored. The original line 100 is renumbered to 101 and the number 100 appears in the current line.

Deleting and Recalling Lines

Lines can be deleted one at a time or in blocks. The [Delete line] ([DEL LN]) key deletes the current line, after which the lower part of the display is refreshed to "close" the space created by the deletion.

Before deletion:

```
90   PRINT "Line 90."
100  PRINT "Line 100."
110  PRINT "Line 110."
```

Make this the current line, then press [Delete line].

After deletion:

```
90   PRINT "Line 90."
110  PRINT "Line 110."
```

New "current line."

If you press the [Delete line] by mistake, you can recover the line by pressing [RECALL] ([f8] in the System menu, or left-most unlabeled key above the numeric keypad, on an ITF keyboard), and then store it by pressing [Return]. The system has a recall buffer that holds the last lines entered, deleted, or executed. You can cycle through these lines, most recent to less recent, by repeatedly pressing the [RECALL] key. [Shift]-[RECALL] cycles through from the current line to the more recent lines. (You can also clear this recall buffer by executing SCRATCH R; this is useful, for instance, when you want to keep others from seeing passwords that you may have typed at the keyboard while accessing protected files.)

When the keyword DEL is followed by a single line identifier, only a single line is deleted. The line identifier can be a line number or a line label. The [Delete line] key produces the same results, but has some advantages.

- You can see the line before you delete it.
- The [Delete line] key saves the line in the recall buffer (the DEL command does not).

Therefore, DEL is more useful for deleting blocks of lines (described in the subsequent section called "Deleting Multiple Lines.")

Copying Lines (By Changing Line Numbers)

Although the computer supplies a line number automatically, you are not forced to use that number if you don't want to. To change the line number, simply back up the cursor and type in the line number you want to use. You can do this to existing lines as a way of copying them to another part of the program. (Note that there is an easier way to copy program lines – by using the COPYLINES command – as described in the next section.)

When you change a line number and store the line, the program is automatically scrolled so that the line just stored is one line above the current-line position. In other words, when you copy a line to a new location, the new location is displayed.

Here are some points to keep in mind when changing the line numbers supplied by the system.

- Changing the line number of an existing line causes a copy operation, not a move. The line still exists in its original location.
- Existing lines are replaced by any line entered with their same line number.
- Be careful that you don't accidentally replace a line because of a typing mistake in the line number.

Getting Out of EDIT Mode

There are many ways to terminate the EDIT mode. Your choice depends upon what you want to do next. If you simply want to return the CRT to its "normal" mode (input line on the bottom and printout area above), press [PAUSE] ([Stop]) or [CLR SCR] ([Clear display]). Any of these keys terminates EDIT mode and returns the screen to the normal format.

Another way to leave EDIT mode is to proceed with another operation. The most useful choices in this case are LIST, CAT, [RESET], [RUN] ([f3] in the System or a User menu of ITF keyboards), or [STEP] ([f1] in the System menu of ITF keyboards). EDIT mode is also terminated by a GET or LOAD operation, and by any operation that uses the display (like LIST and CAT).

Listing the Program

List the program by executing the following command[1] (which also gets you out of EDIT mode, if you have not already terminated it in one of the other ways described in the preceding section):

```
LIST [Return]
```

The system lists the program on the screen (or whichever device is the current PRINTER IS device).

```
10  DISP "Short program."
20  END
```

Storing the Program

You can store the above program, for instance, in the file named "MyProg" on the default volume by executing this statement:

```
STORE "MyProg" [Return]
```

You can also use the SAVE statement, which stores the line in an ASCII representation (rather than in an "internal" representation; see "A Closer Look at Storing Programs" for details.)

Running the Program

Now run the program by typing:

```
RUN [Return]
```

or pressing the [RUN] key ([f3] in the System or in a User menu of ITF keyboards).

The computer should display the following message on the CRT display (or current system display device):

```
Short program.
```

Further information about running programs in described in the "Loading and Running Programs" chapter.

The next sections describe additional editor and system features.

[1] The EDIT binary must be loaded in order to use LIST and EDIT.

A Closer Look at Editing

This section provides a closer look at the BASIC editor[1], showing you more of its powerful and easy-to-use features.

More Details about Getting into EDIT Mode

The EDIT command allows two parameters. The first is a line identifier and the second is the increment between line numbers.

For example, the following command tells the computer to place the program on the CRT so that line 140 is in the current-line position.

```
EDIT 140,20
```

Also, any lines that are added to the program get a line number 20 greater than the previous line.

If the increment parameter is not specified, the computer assumes a value of 10. Thus, the following command tells the computer to place the program on the CRT so that line 1000 is in the current-line position, and added lines get a line number 10 greater than the previous line.

```
EDIT 1000
```

When the line identifier is not supplied, the computer has some interesting ways of assuming a line number.

- If this is the first EDIT after a power-up, SCRATCH, SCRATCH A, or LOAD, the assumed line number is 10.
- If EDIT is performed immediately after a program has paused because of an error, the number of the line that generated the error is assumed.
- At any other time, EDIT assumes the number of the line that was being edited the last time you were in EDIT mode.

[1] The EDIT binary must be loaded in order to use the BASIC editor.

The line identifier also can be a line label. This makes it very easy to find a specific program segment without needing to remember its line number. For example, assume that you want to edit a sorting routine that begins with a line labeled `Go_sort`. Simply type:

```
EDIT GO_SORT
```

The line labeled `Go_sort` is placed in the middle of the display, and lines before and after this line (if any) are displayed above and below this line, respectively.

The EDIT command is not programmable, and you cannot use EDIT mode while a program is running.

In order to locate a program line in a subprogram context, you can use the FIND command. See the subsequent section called "Global Editing Operations" for details.

Typing-Aid Softkey Menu Changes (ITF Keyboards Only)

When you go from "normal" mode to EDIT mode on a system with an ITF keyboard, the softkey menu and labels change to the User 2 menu (if the PDEV and KBD binaries are currently loaded).

While in EDIT mode on an ITF keyboard, however, you can switch softkey menus normally: use either the Shift-Menu key, or the appropriate statements (such as SYSTEM KEYS and USER 1 KEYS) to switch to other menus.

If you are in the User 2 menu when you exit EDIT mode, the system will return you to the menu that was in effect when you entered EDIT mode.

A Closer Look at Listing a Program

All or part of your program can be displayed or printed by executing a LIST statement[1]. The LIST statement allows parameters that specify both the range of lines to be listed and the device to which the listing should be sent.

If the keyword LIST is executed without any parameters, the assumed action is to list the entire program on the system printer.

```
LIST
```

The default system printer after a power-on or SCRATCH A is the CRT. (The system printer is defined by the PRINTER IS statement.)

Starting and ending line numbers can be specified in the LIST statement. For example, the following command lists lines 100 through 200, inclusively.

```
LIST 100,200
```

The following example lists the last portion of the program, from line 1850 to the end.

```
LIST 1850
```

The line identifiers can also be labels. For instance, the following command lists the program from the line labeled "Rocket" to the end.

```
LIST Rocket
```

[1] The EDIT binary must be loaded in order to use the LIST statement and not get the message **(Requires EDIT binary)**.

Directing the listing to a device other than the CRT is easy, but involves concepts that have not been introduced yet. If you want a listing on a printer, you have two choices:

- Specify a different system printer, and then use the LIST statement. For example:

```
PRINTER IS 701
LIST
```

  The parameter 701 identifies the printer connected to the computer through the interface at select code 7 (the built-in HP-IB); the printer itself has an address setting of 01. You can also use the PRT function, which returns a value of 701.

  However, it is often desirable to keep the CRT display as the system printer and still get program listings on an external printer.

- Specifying the printer in the LIST statement. For example, the following command sends the entire program listing to an HP-IB printer (address 01) without changing the system printer selection.

```
LIST #701
```

  When both the printer and the line range are specified, the printer number is specified first and terminated with a semicolon. For example, this command lists lines 200 through 500 on the device connected to the interface at select code 12.

```
LIST #12; 200,500
```

Global Editing Operations

The preceding sections showed how to edit single program lines. This section shows how to perform editing operations that may affect the entire program.

BASIC Command	Purpose and Example Command	Typing-Aid Softkey[1]
REN	renumbers the program (or a specified segment of the program) `REN 100,10`	f1 (k1)
INDENT	indents lines in a program to show the nesting of the branching constructs (such as FOR..NEXT and REPEAT..UNTIL). `INDENT 7,2`	f8 (k4)
FIND	searches the program for a specific textual pattern `FIND "a pattern"`	f6 (k2)
CHANGE	searches for a textual pattern, but allows you to optionally change it to a new pattern `CHANGE "old text" TO "NEW CHARACTERS"`	f7 (k3)
COPYLINES	copies (duplicate) program line(s) to another location in a program. `COPYLINES 10,300 TO 550`	f5
MOVELINES	moves program line(s) to another location in a program. `MOVELINES 450,522 TO 10`	f4
DEL	deletes program segments (ranges of program lines) `DEL 100,150`	Not on a default typing-aid softkey.

[1] Note that the ITF softkeys (f1, f2, etc.) are in the User 2 menu.

This section explains how to use these commands.

Renumbering a Program

After an editing session with many deletes and inserts, the appearance of your program can be improved by renumbering. This also helps make room for long inserts. Renumber programs with the REN command.

This example renumbers the entire program in memory, using a new beginning number of 10 and incremental line numbers of 10:

```
REN
```

Both the starting line number and the interval between lines can be specified. For example, the following example renumbers the entire program, using 100 for the first line number and an increment of 5.

```
REN 100,5
```

If the increment (second parameter) is not specified, 10 is assumed. For example, the command below renumbers the entire program, using 1000 for the first line number and an increment of 10.

```
REN 1000
```

As shown in the first example above, a value of 10 is assumed for starting-line number and line-number increment when no parameters are specified.

You can also renumber only a specified portion of a program. For example, the following command renumbers only line numbers in the range 1000 to 2000:

```
REN  1000,10 IN 1000,2000
```

Indenting a Program

INDENT is also a non-programmable command. It is used to scan an entire program and indent it so as to show the "nesting" of program segments[1] that define:

- Looping (such as FOR..NEXT and REPEAT..UNTIL)
- Conditional execution (such as IF..THEN and SELECT..CASE..END CASE)
- A separate program segment (such as SUB subprograms and DEF FN user-defined functions)

[1] A complete list of the statements that define these constructs is provided in the *BASIC Language Reference* description of the INDENT command.

The following program shows the indentation performed by this command:

```
INDENT 7,2
```

the first parameter (7) indicates the indentation of the "outermost" program segment, and the second parameter (2) shows how many additional spaces each subsequently nested segment is indented. Notice how easy it is to follow the logic flow?

```
10    FOR I=1 TO 5
20      REPEAT
30        INPUT "How old are you?",Age
40        Reasonable=1  ! Assume they're telling the truth...
50        IF Age<0 THEN
60          DISP "A negative age implies you are not born."
70          Reasonable=0
80        ELSE
90          IF Age>120 THEN
100           DISP "Are you sure?!"
110           Reasonable=0
120         ELSE
130           IF Age>100 THEN
140             DISP "You are pretty spry!"
150           ELSE
160             IF Age>80 THEN
170               DISP "Wow! Most people your age don't use computers much."
180             ELSE
190               DISP "Glad to meet you."
200             END IF
210           END IF
220         END IF
230       END IF
240       WAIT 4
250     UNTIL Reasonable
260     DISP "You were";Age*365.2422;" days old on your last birthday."
270     WAIT 3
280   NEXT I
290   END
```

Here is another example of indenting the same program, but with different parameters:

```
   INDENT 5,4

10  FOR I=1 TO 5
20      REPEAT
30          INPUT "How old are you?",Age
40          Reasonable=1  ! Assume they're telling the truth...
50          IF Age<0 THEN
60              DISP "A negative age implies you are not born."
70              Reasonable=0
80          ELSE
90              IF Age>120 THEN
100                 DISP "Are you sure?!"
110                 Reasonable=0
120             ELSE
130                 IF Age>100 THEN
140                     DISP "You are pretty spry!"
150                 ELSE
160                     IF Age>80 THEN
170                         DISP "Wow! Most people your age don't use computers
much."
180                     ELSE
190                         DISP "Glad to meet you."
200                     END IF
210                 END IF
220             END IF
230         END IF
240         WAIT 4
250     UNTIL Reasonable
260     DISP "You were";Age*365.2422;" days old on your last birthday."
270     WAIT 3
280 NEXT I
290 END
```

Indentation Bounds

When indentation parameters attempt to force program statements too far to the right, they are bounded by the width of the screen minus 8 characters. That is, a program line will never start to the right of 8 characters from the right-hand edge of the screen. For instance, on an 80-column screen, a program line will never start to the right of column 72. Instead, all lines which should be indented farther to the right of this column will begin in this column; indentation remains in this column until the nesting level gets back to a manageable point, at which time the line beginnings will begin to drop back to the left.

Removing Indentation

To remove all indenting, execute this command:

```
INDENT 7,0
```

Finding Textual Patterns

When programs are larger than a couple of screenfuls, it is handy to have the computer search for a variable name, numeric or string literal, comment, etc. The non-programmable FIND command allows you to do this.

The following example searches the current program, beginning at the line currently being edited, for the letters "A pattern":

```
FIND "A pattern" [Return]
```

These letters may be a variable name, a string or numeric literal, or a comment (or a portion of any of these).

If you want to begin the search in a different place, then specify the range of lines to be searched:

```
FIND "A pattern" IN 200,650 [Return]
```

Upon executing this command, BASIC begins a search for these characters. The following message is shown in the message/results line (below the "keyboard input line", which is near the middle of the screen in EDIT mode):

```
Finding "A pattern"
```

If the pattern is *not found*, then the system displays the following message:

```
"A pattern" not found
```

If an occurrence of these letters *is found*, the system displays the program line containing the pattern and a confirmation:

```
300   PRINT "A pattern of circles is shown on the display."

Found "A pattern"
```

You can choose any of the following actions:

- Edit the line (optional): move the cursor, and change, add, or delete characters.
- Press [Return] to store the edited (or unchanged) line.
- Scroll the program up or down (with the [▲] or [▼] cursor keys), which cancels the FIND mode.
- Press [CONTINUE] ([f2]) to leave the line unchanged and continue the search.

If you choose to remain in FIND mode, press [Return]. After checking syntax of the line, the FIND command will begin searching for the next occurrence of the specified characters; if the modified line contains a syntax error, you may correct the error and press [Return] again. Once the line is syntactically correct, the FIND command begins searching for the next occurrence of the specified string.

You will remain in FIND mode as long as the FIND command has additional program lines to search. The system reminds you that you are in this mode by displaying these prompts at the bottom, right-hand corner of the screen:

If you want to abort the FIND command, then use the [Break] ([CLR I/O]) key to cancel the mode. The system will display:

```
Search aborted at nnnnn; "A pattern" not found.
```

in which *nnnnn* is the line number at which the FIND was aborted.

Search-and-Replace Operations

The CHANGE command is similar to FIND, except that you will specify both a search pattern and a replacement pattern.

The following example searches for the pattern "Old text" and replaces it with "New characters":

```
CHANGE "Old text" TO "New characters"  Return .
```

Like with FIND, the system shows that it is busy searching for a pattern:

```
Finding "Old text"
```

Also like FIND, the CHANGE command pauses when it finds the first occurrence of the search pattern; however, CHANGE also replaces the old pattern with the new one, and awaits your confirmation/rejection of the change:

```
200   PRINT "New characters."

"Old text" to "New characters"?
```

- Like FIND, you can edit the line first. To confirm the change, press Return .
- To reject the change, press CONTINUE (f2 in the System menu of an ITF keyboard).

If you want only the occurrences of the pattern in a certain program segment to be changed, then use the following syntax:

```
CHANGE "old" TO "New" IN 1, 250
```

If you want all occurrences of the pattern changed, with no capability of interactively confirming/rejecting the changes, use the following syntax:

```
CHANGE "old" TO "New" ; ALL
```

You can also combine these two specifications to change all occurrences within a range of lines:

```
CHANGE "old" TO "New" IN 1, 250; ALL
```

Copying Program Segments

While programming, you may encounter a need to duplicate several lines of BASIC code in another location of the program. With this BASIC system, the COPYLINES command provides an easy way of doing this kind of operation. Make sure the line to which you copy is **not** an already existing line.

The first example below copies lines 180 through 220 to a location beginning at line 5205:

```
COPYLINES 180,220 TO 5205 [Return].
```

The following example copies lines 300 through 3005 to a location beginning at line 100:

```
COPYLINES 300,3005 TO 100 [Return].
```

If the line you try to copy to already exists, an error occurs and no lines are copied. You cannot copy to an already existing line.

Moving Program Segments

Earlier sections of this chapter showed how to use the [RECALL] key to recall a line which was previously deleted. You could use this technique to move one or two lines from one location in a program to another. However, when you are moving several program lines at a time—to a spot several screens away from the original location—there is an easier way: use the MOVELINES command. Make sure the line to which you move is **not** an already existing line.

This example command moves lines 32 through 127, inclusive, to a spot beginning at line 453:

```
MOVELINES 32,127 TO 453 [Return].
```

The following example moves lines 300 through 3005 to a location beginning at line 100:

```
MOVELINES 300,3005 TO 100 [Return].
```

If the line you try to move to already exists, an error occurs and no lines are moved. You cannot move to an already existing line.

Moving Lines into a Subprogram

One of the more frequent uses you may find for the MOVELINES command is in moving program lines from a "main context" into a separate "subprogram context" (defined by SUB and SUBEND statements). However, you may have noticed that to do so you must go to a line *below* all of the existing lines in memory and enter the SUB statement.

```
2100   SUBEND
2110   SUB New_subprogram
2120   _
```

After typing in this subprogram heading, you can use MOVELINES to move program lines from the main program (or from another subprogram) into the new subprogram:

```
MOVELINES 350,499 TO 2120
```

Don't forget to delimit the end of the new context with a SUBEND statement!

```
2630   SUBEND
2640   _
```

Deleting Multiple Lines

The DEL command, introduced in a preceding section, can also be used to delete several lines in a single operation. Blocks of program lines can be deleted by using two line identifiers in the DEL command.

- The first number or label identifies the start of the block to be deleted.
- The second number or label identifies the end of the block to be deleted.

The line identifiers must appear in the same order they do in the program. Here are some examples.

The following command deletes lines 100 through 200, inclusively.

```
DEL 100,200
```

This command deletes all the lines from the one labeled "Block2" to the end of the program.

```
DEL Block2,32766
```

This command would do nothing except generate an error:

```
DEL 250,10
```

If you have subprograms or user-defined functions in your program, they can only be deleted in certain ways (such as with DELSUB). Primarily, the SUB or DEF FN statement cannot be deleted without deleting the entire subprogram or function. This subject is explained fully in the "Subprograms" chapter of *BASIC Programming Techniques.*

The DEL command is not programmable and cannot be used while a program is running.

Making Programs Readable

When first learning how to program, most people view the use of comments, long variable names, descriptive printouts, and other documentation tools as merely extra typing that isn't really necessary in their short programs. As time passes, old programs are expanded, new programs are written, and more people use the program. Eventually, software support activities become necessary. Some obscure bug is found or some exciting enhancement is requested. The programmer picks up a copy of a program written a year ago and can't begin to remember what "X1" was or why you would ever want to divide it by "X2". Program documentation can make the difference between a supportable tool that adapts to the needs of the users and a support nightmare that never really does exactly what the current user wants. Keep in mind that the local software support person just might be you.

This BASIC language makes it easy to write self-documenting programs. In addition to BASIC's standard REM (remark) statement, additional documentation features are:

- Descriptive keywords (such as REPEAT..UNTIL, LOOP..END LOOP, and so forth)
- Descriptive variable names (up to 15 characters)
- Descriptive line labels (up to 15 characters)
- End-of-line comments.

Contrast Between Documented and Undocumented Programs

Although this section deals primarily with commenting methods, all of these features work together to make a readable program. The following example shows two versions of the same program. The first version is uncommented and uses "traditional" BASIC variable names. The second version uses the features of HP's BASIC language to make the program more easily understood. Which version would you rather work with?

```
100  PRINTER IS 1
110  A=.03
120  B=.02
130  X=0
140  Y=0
150  C=A+B
160  PRINT "  Item       Total      Total"
170  PRINT "  Price       Tax       Cost"
180  PRINT "------------------------------"
190  P=0
200  INPUT "Input item price",P
210  D=P*C
220  E=P+D
230  X=X+D
240  Y=Y+E
250  DISP "Tax =";D;"Item cost =";E
260  PRINT P,X,Y
270  GOTO 190
280  END
```

```
100  ! This program computes the sales tax for
110  ! a list of prices. Item prices are input
120  ! individually. The tax and total cost for
130  ! each item are displayed. The running
140  ! totals for tax and cost are printed on
150  ! the CRT. Modify line 220 to change the
160  ! the system printer.
170  !
180  ! Sales tax rates are assigned on lines 230
190  ! and 240. The rates used in this version
200  ! of the program were in effect 1/1/81.
210  !
220  PRINTER IS CRT          ! Use CRT for printout
230  State_tax=.03           ! Local tax rates
240  City_tax=.02
250  !
260  Total_tax=0             ! Initialize variables
270  Total_cost=0
280  Tax_rate=State_tax+City_tax
290  ! Print column headers
300  PRINT "  Item      Total      Total"
310  PRINT "  Price      Tax       Cost"
320  PRINT "------------------------------"
330  !
340 LOOP  ! Start of main "Get Price" loop.
350    Price=0       ! Don't change totals if no entry.
360    INPUT "Input item price",Price
370    Tax=Price*Tax_rate
380    Item_cost=Price+Tax
390    Total_tax=Total_tax+Tax   ! Accumulate totals.
400    Total_cost=Total_cost+Item_cost
410    DISP "Tax =";Tax;" Item cost =";Item_cost
420    PRINT Price,Total_tax,Total_cost
430  END LOOP        ! Repeat loop for next item.
440  END
```

There are two methods for including comments in your programs. The use of an exclamation point is demonstrated in the second example program. The exclamation point marks the boundary between an executable statement and comment text. There does not have to be an executable statement on a line containing a comment. Therefore, the exclamation point can be used to introduce a line of comments, to add comments to a statement, or simply to create a "blank" line to separate program segments. Exclamation points may be indented as necessary to help keep the comments neat.

The REM statement can also be used for comments. The exclamation point is neater and more flexible, but the REM statement provides compatibility with other BASIC languages. The REM keyword must be the first entry after the line identifier and must be followed by at least one blank. Note also that the REM statement moves when a program is indented (with INDENT); however, "!" comments do not get indentation changed (unless forced by other text in that line).

Here are some examples of proper and improper REM statements and "!" comments.

```
        Right                          Wrong
----------------------------  -------------------------------

10 REM Check Book Balance      20 REMinitialize array

40 Start2: ! Subtotal loop     50 X=PI*R^2  REM Area of circle
```

General Recommendations for Commenting Programs

Each programmer has an individual style in the use of comments. Therefore, the following is not a list of rules. It is simply some suggestions on the effective use of comments.

1. Include a **program heading** that answers the following questions:
   a. Why was the program written?
   b. What does it do?
   c. Who would probably use it?
   d. Who is the author of the program?
   e. When was the date of the last revision?
   f. Who is currently in charge of supporting it?
   g. What modifications could/should be made by a normal user?
2. **Describe all variables**, especially global variables. A descriptive variable name may do the job, or a more detailed explanation may be needed.
3. Describe any **hardware or software configuration** required for the proper running of the program. This may even include an explanation of how to modify the program to accommodate alternate devices (when such changes are reasonable).
4. **Make major blocks and entry points visible.** Many tools are available for this, including descriptive labels, indenting, spacing, and comments describing program flow.

5. **Use comments freely** to describe the action of complex lines, equations, fancy manipulations, and "low-level" operations like CONTROL statements and escape code sequences. These heavily coded operations can be very important to the computer but very mysterious to the human trying to read the program.

Software Security

There may be times when you want to keep portions of your programs from being read or used by other programmers or users. With this BASIC system, and Series 200/300 computer hardware, you can either prevent a program from being read, or from being executed unless you give the authorization.

Preventing Programs from Being Listed

With this BASIC system, you can use the SECURE statement to prevent program line(s) from being listed. (Another way is to simply make sure that the EDIT binary is not loaded in the system or available to anyone who you don't want to look at *any* programs. However, that is not a highly restrictive method since the EDIT binary is available as a standard component of most Series 200/300 BASIC systems.)

The following example secures lines 30 through 60 from being listed (either with the editor or by using the LIST statement):

```
SECURE 30,60
```

Here is what the program might look like—either with the editor or as the output of a LIST statement:

```
10    ! Example of SECURE'd program.
20    ! Begin password check routine.
30*
40*
50*
60*
70    ! End of password check.
80      .
        .
        .
```

If you want the whole program to be secured, use this statement:

```
SECURE
```

Note

Once a program is secured, it **cannot** be un-secured. Therefore, you should keep an un-secured back-up copy of all programs.

Other Security Measures

There is also another method of preventing software from being used by anyone who may acquire a copy of it. You can write a routine that checks the serial number of a computer, or the serial number of an optional HP 46084 ID Module. Then your routine can determine whether or not to permit the rest of the program to be executed on this hardware configuration.

Reading an ID PROM

To read the serial number of a computer, use the following statement:

```
SYSTEM$("SERIAL NUMBER")
```

The function returns the contents of the ID PROM, if present, or the null string if no ID PROM is present.

Reading an ID Module's Contents

The same function:

```
SYSTEM$("SERIAL NUMBER")
```

also returns the **encoded** contents of an ID Module. In order to decode an ID Module's contents, use the "ID_MODULE" program supplied on the *Manual Examples* disc.

A Closer Look at Storing Programs

To write a program to a mass storage device, you will use either the SAVE or the STORE statement. There is no "right" or "wrong" choice; your choice depends upon the kind of file you want.

- STORE records an internal representation of the program in a PROG file. The main advantage of a PROG file is a rapid retrieval rate.
- SAVE[1] records the actual text of the program in an ASCII file. The main advantage of an ASCII file is that it can be read as data by a BASIC program or by LIF-compatible[2] devices (such as other HP computers and terminals).

The following table gives a summary of the differences between SAVE and STORE.

Characteristic	SAVE	STORE
File type created:	ASCII	PROG
Retrieved by:	GET	LOAD
Approximate storage speed:	900 bytes/s	13 000 bytes/s[3]
Approximate retrieval speed:	300 bytes/s[4]	14 000 bytes/s[3]
Can file be read as data?	Yes	No
LIF-compatible file?	Yes	No
Arbitrary program segments allowed?	Yes	No
Subprograms included?	Yes	Yes
Can use LOADSUB to retrieve a subprogram?	No	Yes
Stores CSUBs correctly	No	Yes

[1] Using SAVE requires the EDIT binary.

[2] LIF is the acronym for Logical Interchange Format, which is a disc format used by several HP divisions. (Note that the first letter of the file name must be a letter; in addition, some LIF-compatible devices restrict file names to upper-case letters and the decimal digits 0 through 9.)

[3] The speeds for LOAD and STORE are approximate for an interleave factor of 1 on an HP 9836 internal disc drive. Interleave factors greater than this will cause a corresponding decrease in speed.

[4] The retrieval speed for GET is very data-dependent. On an HP 9836 with a clock rate of 8 MHz, for instance, it can vary from 20 bytes/second to 600 bytes/second (and maybe beyond those limits) according to the contents of the file and the syntax checking required to enter the lines into program memory. See the "Efficient Use of the Computer's Resources" for a discussion of how to time various computer operations.

Using STORE

The following command creates a program file called "MyProgFile" on the current default volume:

```
STORE "MyProgFile".
```

If you to get error 54, that means that there is already a file on the disc with the name you are using. In this event, you have three choices.

- Pick a name that doesn't already exist. To determine which file names are already being used, execute a CAT command.
- You may want to replace the existing file with a new one (like when you update program files with a new, improved version). To replace an existing file, use the RE-STORE statement. For example, the command to replace a program file called "BEAMS" is:

  ```
  RE-STORE "BEAMS"
  ```

  Note that the hyphen must be used in the RE-STORE statement. (RESTORE without a hyphen is used to reset the pointer associated with DATA statements: see the "Data Storage and Retrieval" chapter of *BASIC Programming Techniques* for details.)
- PURGE the old file, then STORE the new one.

Using SAVE

The SAVE operation is similar to the STORE operation in that it stores the current program in a file; however, it has one additional feature. The SAVE statement allows line identifiers that specify what portion of the program you want to save. This is especially helpful when moving or appending program segments during major editing operations. Here are some examples of using the SAVE statement.

To save all of a program in an ASCII file called "WHALES", on the current default volume, execute the following command:

```
SAVE "WHALES"
```

The following command saves the last part of a program, from line 500 to the end, in an ASCII file called "LastPart".

```
SAVE "LastPart",500
```

When both the starting and ending lines are specified, any arbitrary portion of a program can be saved. Executing the following command saves that portion of a program that is between the lines labeled "Sort" and "Printout" (inclusive) in an ASCII file called "Sorter".

```
SAVE "Sorter",Sort,Printout
```

There is also a RE-SAVE statement that allows an existing file to be replaced by a newly created file with the same name. For example, to update an ASCII file called "Analysis" with a new version of the program, the following command would be used.

```
RE-SAVE "Analysis"
```

Creating Files Compatible with vi

If you want to create a file that you can edit with the HP-UX system's *vi* editor, here are the steps to take:

1. Use the CREATE statement to create a file of type HP-UX:

   ```
   CREATE "ux_file",1
   ```
   (number of records is not important)

2. Use the RE-SAVE statement to save the program in the file you just created:

   ```
   RE-SAVE "ux_file"
   ```

   The reason for the RE-SAVE (instead of just SAVE) is that the BASIC system would have created an ASCII type file with SAVE, while the RE-SAVE will maintain the file type (which in this case is a file of type HP-UX).

What to Do Next

Task/Topic	Chapter/Section
Learn how to load and run programs.	"Loading and Running Programs"
Learn how to use and manage files.	"Using Files and Directories"
Learn about each key on your keyboard.	"Keyboard Reference"
Learn how to maintain your system.	*Maintaining BASIC* section, *Installing and Maintaining the Basic System*
Learn about utilities available with the BASIC system.	"BASIC Utilities Library", *Installing and Maintaining the Basic System*

Task/Topic	Where to Go for Details
BASIC ITF keyboard overlays	7-2
Character enter keys	7-2
Cursor-control keys	7-5
Numeric keypad	7-6
Editing keys	7-7
Program control keys	7-9
System control keys	7-10
Softkeys and softkey control	7-12

ITF Keyboards 7

If you do not have the ITF keyboard, skip ahead to one of the following chapters, which describe the HP 98203B/C and HP 98203A keyboards.

The keys on the ITF keyboard are arranged into the following functional groups:

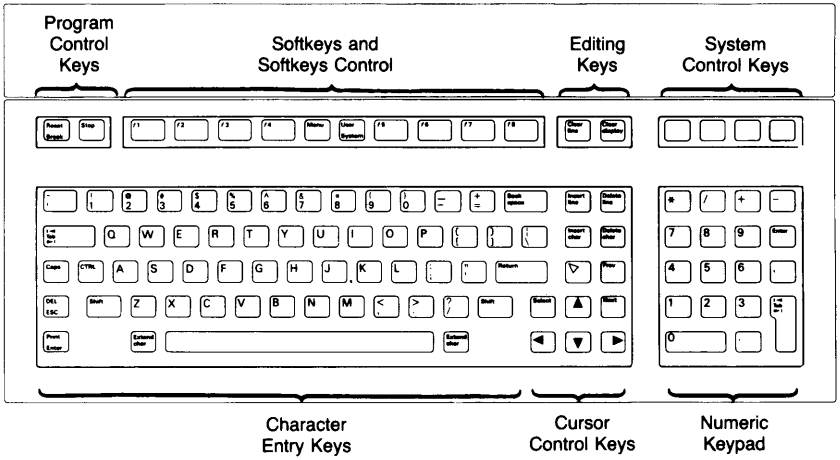


Figure 7-1. ITF Keyboard

This chapter provides a handy reference guide to BASIC's key definitions for the ITF keyboard. Keep in mind that other system programs may define the keys differently. Each key will be demonstrated where possible. One point to clarify: the **cursor** that we refer to in the following paragraphs is the blinking-underline that points to a location on the screen. (If you have a Model 237 computer, an HP 98700 Graphics Display station, or an HP 98548A, HP 98549A, or HP 98550 display, the cursor does not blink.)

Note

Before you proceed, type:

`SCRATCH` Return

This clears the computer of any programs that might be left in memory from previous demonstrations.

BASIC ITF Keyboard Overlays

Two keyboard overlays designed for the ITF keyboard were included with your BASIC Language System. Place the overlays on the keyboard as shown below:

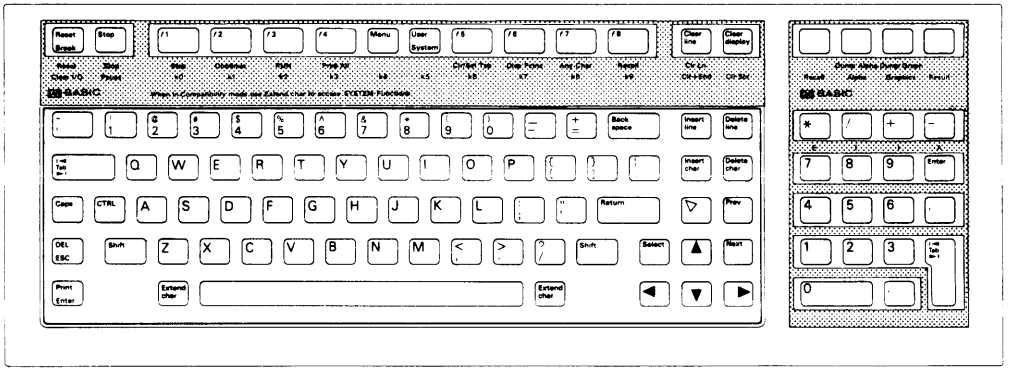
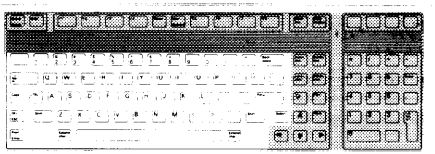


Figure 7-2. BASIC Keyboard Overlays

Character Entry Keys



The character entry keys are arranged like a typewriter, but have some added features.

Caps

The Caps key sets the unshifted keyboard to either upper-case (which is the default after BASIC is booted) or lower-case (normal typewriter operation). The computer displays which mode the computer is in when you press the Caps key.

Type a few words, then press Caps and continue typing. Notice the case change. Press Shift-Clear line when finished.

[Shift]

You can enter standard upper-case and lower-case letters, using the [Shift] key to access the alternate case.

Type a few words, pressing [Shift] to change the case of the first letter of each word. Now press [Caps] and continue typing. Notice that the alternate case accessed by [Shift] depends on the setting of [Caps]. Press [Shift]-[Clear line] when finished.

[Return]

The [Return] key has three functions:

- When a running program prompts you for data, respond by typing the requested data and then pressing [Return]. This signals the program that you have provided the data and that it can resume execution.
- When typing in lines of a program, the [Return] key is used to store each line of program code.
- After typing in a command, the [Return] key causes the command to be executed.

Type `EDIT` and press [Return]. Notice the number `10` now displayed on the screen—this is the line number of the first line of a BASIC program. The computer is waiting for you to type in the line. Type:

```
!FIRST LINE
```

and press [Return]. Notice that the computer accepts the statement as a program line and displays `20` in preparation for the next one. Press [Stop] when finished.

[Enter]

Pressing [Enter] is the same as pressing the [Return] key.

[Print]

Pressing [Print] ([Shift]-[Enter]) prints a complete copy of the alpha display on the default printer. The shifted version of the key directly above the [/] key in the numeric keypad (labeled Dump Alpha on the overlay) performs the same function.

Extend char

When pressed along with another key, this key allows you to generate the rest of the full 256-bit character set from the main typewriter section on Standard and European keyboards (see illustration). On a Katakana keyboard, the "Roman" and "Katakana" keys select the other character sets. To get Katakana characters 161 through 254 on a medium-resolution Series 300 screen, you must load the LEX language extension binary.

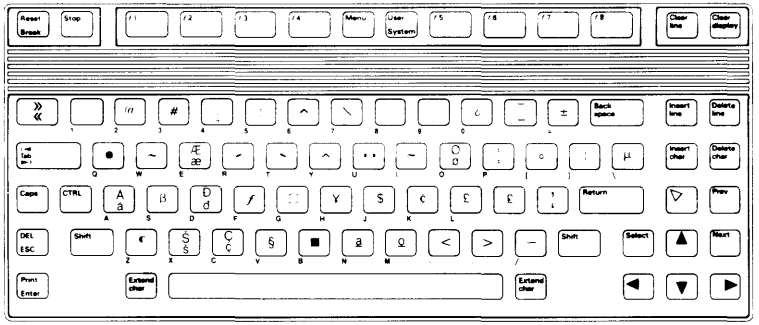


Figure 7-3. Extended Character Set

Tab

The Tab key moves the cursor forward to preset tabs. Pressing Shift-Tab moves the cursor backward to preset tabs.

Before Tab can be used, a tab must be set. Tabs are set and cleared with System menu softkeys. The Tab key is demonstrated along with the **Set Tab/Clr Tab** softkey under "System Softkeys" later in this chapter.

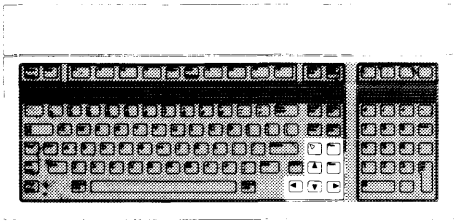
CTRL

The CTRL (control) key works like Shift to access a set of standard control characters, such as line-feed and form-feed. These characters are useful to the programmer for controlling some devices and for communicating with other computers. You probably won't need them when running programs. The available control characters are listed in the *BASIC Language Reference* in the "Useful Tables" appendix.

Select

The Select key beeps but performs no function unless it is program-defined.

Cursor-Control Keys



The cursor-control keys move the display cursor. The [▲] and [▼] keys allow you to scroll lines in the output area up and down. Shifted, the keys allow you to "jump" to the top and bottom of the output area. The [►] and [◄] keys allow you to move horizontally along a line. Shifted, they allow you to "jump" to the left and right limits of a line. The [Back space] key works just like the [◄] key.

The unshifted [↖] key positions the print position at the beginning position on the page. The shifted [↖] key places the print position at the beginning of the first empty line in the display (scrolls up if necessary). In edit mode, pressing this key (shifted or unshifted) causes the computer to beep.

To verify operation of the [↖] key, press [Clear display]. Then type `PRINT "SOMETHING"` and press [Return]; repeat twice. You should now have the following display:

```
SOMETHING
SOMETHING
SOMETHING
```

Press the [↖] key (unshifted).

Type `PRINT "ANY " and press` [Return]. Your display should look like this:

```
ANY THING
SOMETHING
SOMETHING
```

Press [Clear display].

In normal mode, pressing the [Prev] key causes the display to scroll down one page and pressing the [Next] key causes the display to scroll up one page. In edit mode, these keys move the display one-half page.

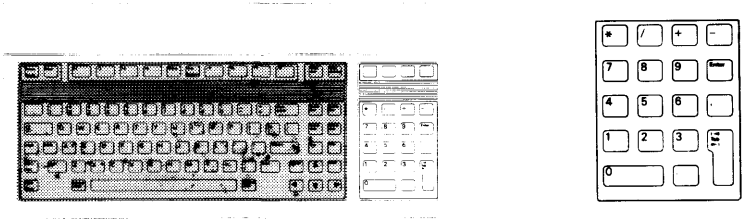
To test the horizontal movement of the cursor, type a few words and press the shifted and unshifted [◄] and [►] keys. Notice that the cursor cannot be moved beyond the characters you have typed. Press [Shift]-[Clear line] when finished.

To test the vertical movement of the cursor, type EDIT and press [Return]. Now type the following lines, pressing [Return] after each line (the first line may be there already, so just press [Return] to accept it):

```
10 !FIRST LINE
20 !SECOND LINE
30 !THIRD LINE
40 !FOURTH LINE
```

Try out the shifted and unshifted [▲], [▼], and [►] keys. Then try the [Prev] and [Next] keys. When you're done, press [Stop] to exit. Then, type SCRATCH [Return] to clear memory.

Numeric Keypad



The numeric keypad provides a convenient way to enter numbers and perform arithmetic operations. Simply type in the arithmetic expression you want to evaluate, then press [Enter]. The result is displayed in the lower-left corner of the screen.

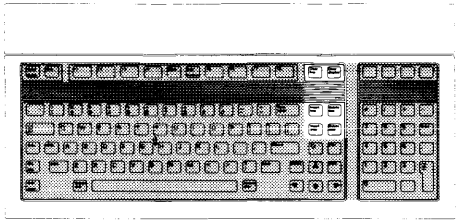
The [Enter] key performs the same function as the [Return] key. The [Tab] key on the numeric keypad functions like the [Tab] key in the character entry area. The shifted versions of the [*], [/], [+], and [-] keys are E, (,), and ^, respectively (see labels on the overlay). The shifted versions are also available in the character entry area.

Type in the following problem using the numeric keypad:

```
(26+14)/4
```

Now press [Enter] to perform the calculation. The answer, 10, is displayed in the lower-left corner of the screen.

Editing Keys



The editing keys put easy character editing and line editing at your fingertips.

[Insert line]

Pressing [Insert line] inserts a new line above the cursor's current position (edit mode only).

Type **EDIT**, then press [Return]. Type in this line (if it isn't already there):

```
10 !FIRST LINE
```

Now, with the cursor somewhere on line 10, press [Insert line]. Notice that a new line number (1) is inserted before line 10. Press [Stop] when finished.

[Delete line]

Pressing [Delete line] deletes the line containing the cursor (edit mode only).

Type **EDIT**, then press [Return]. Position the cursor to the line:

```
10 !FIRST LINE
```

and press [Delete line]. The line is removed. To restore it, press the key directly above [*] (labeled Recall on the overlay) to recall it, then press [Return] to enter it into the program. Press [Stop] to exit edit mode.

[Insert char] Pressing [Insert char] sets insert mode, allowing you to insert characters to the left of the cursor. Press the key a second time to cancel insert mode.

Carefully type the following line exactly as shown:

```
THIS IS A TEST .
```

Position the cursor under the period and press [Insert char]. Now type:

```
OF INSERT MODE
```

and press [Insert char] again. The line should now look like this:

```
THIS IS A TEST OF INSERT MODE.
```

The new characters were inserted to the left of the period. Press [Shift]-[Clear line] when finished.

[Delete char] Pressing [Delete char] deletes the character at the cursor's position.

Type a few words and experiment with [Delete char], positioning the cursor at various places on the line. Notice that if you hold the key down, characters are deleted until you release it. Delete all of the characters you typed.

[Clear line] Pressing unshifted-[Clear line] (labeled Clr → End on the overlay) clears from the current cursor position to the end of the line.

Pressing [Shift]-[Clear line] (labeled Clr Ln on the overlay) clears the keyboard line and message/results line.

Type in a few words and use the [◄] key to position the cursor in the middle of the line. Press unshifted-[Clear line] to clear to the end of the line. Press [Shift]-[Clear line] to clear the rest of the line.

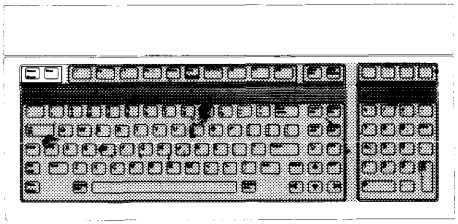
[Clear display] Pressing either the shifted or unshifted version of [Clear display] clears the entire alpha screen.

Type the following BASIC command:

```
PRINT "PUT THIS MESSAGE IN THE OUTPUT AREA."
```

Now press [Return] to execute it. Press the key directly above [*] (labeled Recall on the overlay) to recall the command, and press [Return] again. Repeat this step several times to fill the screen with messages. Now press [Clear display] to erase all lines at once.

Program Control Keys



The following keys allow you to control execution of the program stored in the computer's memory.

[Stop]

Pressing unshifted-[Stop] (labeled Pause on the overlay) **pauses** program execution after the current line. Pressing **Continue** (unshifted [f2]) in the System menu resumes program execution from the point where it was paused.

Pressing [Shift]-[Stop] (labeled Stop on the overlay) **stops** program execution after the current line. To restart the program, press **RUN** (unshifted [f3]) in the System menu.

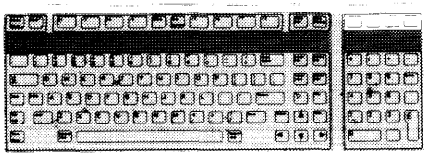
[Break]

Pressing [Break] (labeled Clr I/O on the overlay) pauses program execution when the computer is performing or trying to perform an I/O operation. Press [Break] instead of unshifted-[Stop] when the computer is hung up on an I/O operation, since unshifted-[Stop] works only after the computer finishes the current program line. Pressing [Break] cancels the I/O operation and pauses the program at the current line.

[Reset]

Pressing [Reset] ([Shift]-[Break]) pauses program execution immediately without erasing the program from memory. The `BASIC Reset` message indicates the computer is ready for your command.

System Control Keys



Four unlabeled keys directly above the numeric keypad control various system functions related to the display, printer, and editing operations. Most of these keys execute their functions immediately, as the key is pressed.

To easily identify the keys in the following description, we'll use this convention:

- Key 1—Above the [*] key (labeled Recall on the overlay).
- Key 2—Above the [/] key (labeled Alpha/Dump Alpha on the overlay).
- Key 3—Above the [+] key (labeled Graphics/Dump Graph on the overlay).
- Key 4—Above the [-] key (labeled RES on the overlay).

Key 1—Recall Pressing unshifted-Key 1 (Recall) recalls the last line that you entered, executed, or deleted. Several previous lines can be recalled this way. Recall is particularly handy to use when you mistype a line. Instead of retyping the entire line, you can recall it, edit it using the editing keys, and enter or execute it again.

Type:

```
PRINT "1"  [Return]
```

to print the number 1 on the screen. Now press Key 1 to recall the print statement. Edit the statement to print the number 2 by positioning the cursor under the 1 and typing [2] over it. Press [Return] again. Now press Key 1 several times to see all of the statements it remembers. Then press [Clear display] when finished.

[Shift]-Key 1 moves forward through the recall stack.

Pressing [f8] in the System menu performs the same recall function as Key 1.

Key 2—Alpha/ Dump Alpha

Pressing unshifted-Key 2 (Alpha) once turns on the alphanumeric display. Pressing it the second time turns off the graphics display. This key function requires that the GRAPH BIN file be loaded. If you have a Model 237, an HP 98700 Graphics Display station, or Series 300 computer, this key may perform no function.

Pressing [Shift]-Key 2 (Dump Alpha) prints a complete copy of the alpha display on the default printer. The Dump Alpha function is also executed by [Print].

Key 3—Graphics/ Dump Graph

Pressing unshifted-Key 3 (Graphics) once turns on the graphics display. Pressing it the second time turns off the alphanumeric display. If you have a Model 237, an HP 98700 Graphics Display Station, or Series 300 computer, this key may perform no function.

Pressing [Shift]-Key 3 (Dump Graph) prints a complete copy of the graphics display on the default printer. If you have a Model 237, an HP 98700 Graphics Display Station, or Series 300 computer, the combined alpha and graphics display is printed.

Both key functions require that the GRAPH language extension file be loaded.

Key 4—RES

Pressing Key 4 (RES) either shifted or unshifted returns the result of the last arithmetic expression that was executed.

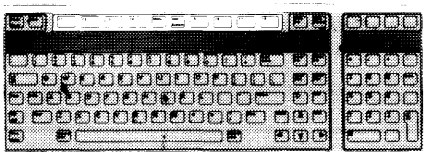
Press [Shift]-[Clear line], then type:

```
23+45 [Return]
```

The result, `68`, is displayed in the lower-left corner of the screen. To add 123 to this value, press Key 4 and type:

```
+123 [Return]
```

The new result, `191`, is now displayed. Press [Shift]-[Clear line] when finished.



There are eight softkeys (labeled [f1] through [f8] and two keys that control the definitions of the softkeys ([Menu] and [System]).

When the BASIC system is booted, the softkeys default to System mode. The System mode menu that appears at the bottom of your display is shown. System softkeys are defined following control key definitions. In addition to the System mode, there are also three User modes: User 1, User 2, and User 3. *BASIC Programming Techniques* describes how to set up User modes.

Softkey Control Keys

[System] — Pressing unshifted-[System] causes softkeys to assume System mode. The System menu is displayed, **if** the [Menu] key is toggled to the "on" position.

[User] — Pressing [User] ([Shift]-[System]) puts the softkeys in User mode. A User menu is displayed **if** the [Menu] key is toggled to the "on" position[1].

[Menu] — Pressing unshifted-[Menu] toggles the softkey labels—turns them on if they're off and turns them off if they're on.

Pressing [Shift]-[Menu] increments User mode and menu **if** User mode is "on".

User menus are blank unless the KBD language extension binary is loaded. After the KBD binary is loaded, the softkeys default to the User 1 mode.

[1] The system remembers which User menu you were in when you press the [System] key and returns to that menu when you press the [User] key. A second press of the [User] key will always go to the User 1 menu. There are additional iterations with EDIT mode; see "Typing Aid Softkey Menu Changes" in the "Editing and Storing Programs" chapter for details.

Let's get familiar with the two control keys.

First we want to get the System mode selected and menu displayed. If the System menu is displayed, continue with the next paragraph. If it is not displayed, press [System]. If it is still not displayed, press [Menu].

With the System menu displayed, press unshifted-[Menu] several times. The system menu display should go on and off. Leave the System menu displayed, and continue.

Now press [Shift]-[User]. The User 1 menu should appear on your display.

Press [Shift]-[Menu] several times. The displayed menus should rotate successively through the three User menus (User 1 → User 2 → User 3 → User 1 → User 2, etc.).

Press unshifted-[Menu] several times and the last User menu goes on and off. Leave the User menu on.

Finish this exercise by pressing unshifted-[System] to get your computer back in System mode.

System Softkeys

The following paragraphs define the eight System softkeys.

Step	**Step** (unshifted-[f1]) allows you to execute one program line at a time. This is particularly useful for debugging (fixing) programs.
Continue	**Continue** (unshifted-[f2]) resumes program execution from the point where it was paused (by an unshifted-[Stop]).
RUN	**RUN** (unshifted-[f3]) starts a program running from the beginning.

Print All

The **Print All** key (unshifted-[f4]) turns the printall mode on and off, allowing keyboard operations and displayed error messages to be copied to a printall device. Press **Print All** once to set printall "on" and again to set printall "off". An asterisk (*) appears next to `All` to indicate that printall is "on".

The display's output area is the default printall device at powerup. *BASIC Programming Techniques* explains how to select other printall devices.

Press **Print All** to turn on printall mode. Now type in the following command:

```
PRINT "THIS IS A KEYBOARD OPERATION"  [Return]
```

Both the PRINT command and the message itself are displayed on the screen, which is the default printall device. Now type:

```
THIS WILL CAUSE AN ERROR  [Return]
```

Because this is not an executable BASIC statement, an error message is displayed, both at the bottom of the screen and in the printall area at the top. This way, a log is produced of all commands typed and executed at the keyboard, along with any error messages. Press [Clear display] to clear the display, and press **Print All** to turn off printall mode.

Set Tab/Clr Tab

Set Tab (unshifted-[f5]) sets a tab at the cursor's current position. Tabs remain in effect until cleared by either **Clr Tab** or the SCRATCH A statement (explained in *BASIC Programming Technique*).

Clr Tab ([Shift]-[f5]) clears a tab previously set at the cursor's position.

Press the space bar to move the cursor forward a few spaces and press **Set Tab**. Move the cursor back several spaces using [◄], then press [Tab]. Move the cursor forward several more spaces with the space bar, then press [Shift]-[Tab]. To clear the tab, move the cursor to the unwanted tab position and press **Clr Tab**. Press [Shift]-[Clear line] when finished.

Display Fctns

Display Fctns (unshifted-[f6]) sets the display-functions mode, allowing you to see special control characters (e.g., form-feed, carriage return) on the screen. Pressing this key a second time cancels the display-functions mode. An asterisk (*) appears next to **Fctns** to indicate that display-functions mode is "on".

Type the following line:

```
PRINT "DISPLAY-FUNCTIONS MODE OFF"  [Return]
```

Notice the display at the top of the screen. Now press **Recall** (unshifted-[f8]) to recall the line, and edit it to read:

```
PRINT "DISPLAY-FUNCTIONS MODE ON"
```

Press **Display Fctns**, and then press [Return]. Notice that the carriage return (CR) and line-feed (LF) control characters are now displayed. Press **Display Fctns** again to exit display-functions mode. Press [Clear display] when finished.

Any char

Any char (unshifted-[f7]) is used to find any ASCII character. First press **Any char**. The following message appears above the menu:

```
Enter 3 digits, 000 to 255
```

Enter a three-digit number from 000 through 255 representing the decimal equivalent of an ASCII character. The computer automatically displays the character on the screen. For a list of characters and their equivalent decimal values, see the US ASCII Character Codes table in the "Useful Tables" appendix of the *BASIC Language Reference.*

Press **Any char**, then type 65 which is the decimal equivalent of "A". The display line now displays "A". Press [Shift]-[Clear line] to erase it.

Recall

The **Recall** softkey (unshifted-[f8]) acts just like System Control Key 1 (described earlier). **Recall** recalls the last line that you entered, executed, or deleted. Several previous lines can be recalled this way. **Recall** is particularly handy to use when you mistype a line. Instead of retyping the entire line, you can recall it, edit it using the editing keys, and enter or execute it again.

Type:

```
PRINT "1"  [Return]
```

to print the number `1` on the screen. Now press **Recall** to recall the PRINT statement. Edit the statement to print the number `2` by positioning the cursor under the `1` and typing [2] over it. Press [Return] again. Now press **Recall** several times to see all of the statements it remembers. Note that **Recall** goes backward through the queue.

Pressing [Shift]-[f8] allows you to cycle forward through the queue until the last line entered, executed, or deleted is displayed. In the previous exercise you pressed unshifted-[f8] several times, cycling backward through the queue. Now press [Shift]-[f8] several times to cycle forward through the queue until the last line is displayed.

Task/Topic	Where to Go for Details
Character entry keys	8-2
Numeric keypad	8-4
Cursor-control keys	8-5
Editing keys	8-6
System control keys	8-9
Softkeys	8-11
Program control keys	8-12

HP 98203B/C Keyboards

8

If you have the ITF keyboard, refer to the preceding chapter. If you have the HP 98203A keyboard, skip to the following chapter.

HP 98203B/C keys are arranged into the following functional groups:

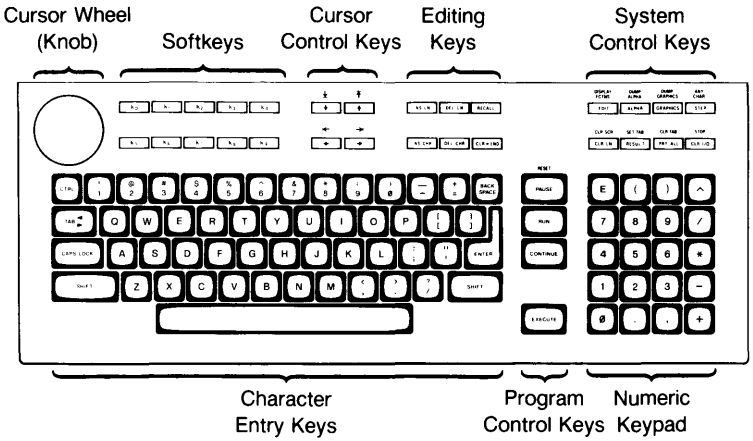


Figure 8-1. HP 98203B/C Keyboard

This chapter provides a handy reference guide to BASIC's key definitions for the HP 98203B/C keyboard. Keep in mind that other system programs may define the keys differently. Each key will be demonstrated where possible. One point to clarify: the **cursor** that we refer to in the following paragraphs is the underline that points to a location on the screen.

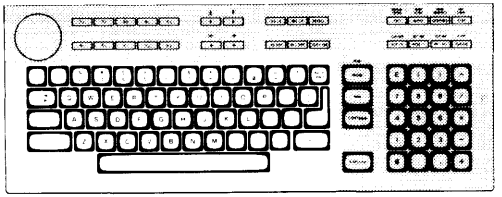
Note

Before you proceed, type:

`SCRATCH` [EXECUTE]

This clears the computer of any programs that might be left in memory from previous demonstrations.

Character Entry Keys



The character entry keys are arranged like a typewriter, but have some added features.

[CAPS LOCK] The [CAPS LOCK] key sets the unshifted keyboard to either upper-case (which is the default after BASIC is booted) or lower-case (normal typewriter operation). The computer displays which mode the computer is in when you press the [CAPS LOCK] key.

Type a few words, then press [CAPS LOCK] and continue typing. Notice the case change. Press [CLR LN] when finished.

[SHIFT] You can enter standard upper-case and lower-case letters, using the [SHIFT] key to access the alternate case.

Type a few words, pressing [SHIFT] to change the case of the first letter of each word. Now press [CAPS LOCK] and continue typing. Notice that the alternate case accessed by [SHIFT] depends on the setting of [CAPS LOCK]. Press [CLR LN] when finished.

[ENTER]

The [ENTER] key has several functions:

- When a running program prompts you for data, respond by typing the requested data and then pressing [ENTER]. This signals the program that you have provided the data and that it can resume execution. The [EXECUTE] key can also be used for this function.
- When typing in lines of a program, the [ENTER] key is used to store each line of program code. The [EXECUTE] key can also be used for this function.
- Like the [EXECUTE] key, the [ENTER] key can be used to execute commands and calculations.

Type `EDIT` and press [ENTER]. Notice the number `10` now displayed on the screen—this is the line number of the first line of a BASIC program. The computer is waiting for you to type in the line. Type:

```
!FIRST LINE
```

and press [ENTER]. Notice that the computer accepts the statement as a program line and displays `20` in preparation for the next one. Press [PAUSE] when finished.

[TAB]

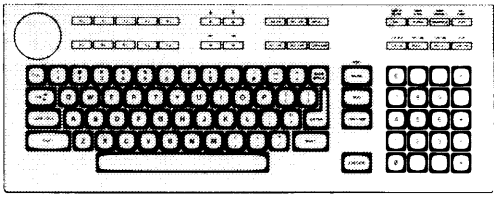
The [TAB] key moves the cursor forward to preset tabs. Pressing [SHIFT]-[TAB] moves the cursor backward to preset tabs.

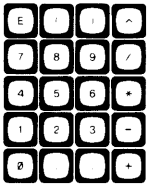
Before [TAB] can be used, a tab must be set. Press the space bar to move the cursor forward a few spaces and press [SET TAB] ([SHIFT]-[RESULT]). Move the cursor back several spaces using [←], then press [TAB]. Move the cursor forward several more spaces with the space bar, then press [SHIFT]-[TAB]. To clear the tab, move the cursor to the unwanted tab position and press [CLR TAB] ([SHIFT]-[PRT ALL]). Press [CLR LN] when finished.

[CTRL]

The [CTRL] (control) key works like [SHIFT] to access a set of standard control characters, such as line-feed and form-feed. These characters are useful to the programmer for controlling some devices and for communicating with other computers. You probably won't need them when running programs. The available control characters are listed in the "Useful Tables" appendix of *BASIC Language Reference.*

Numeric Keypad





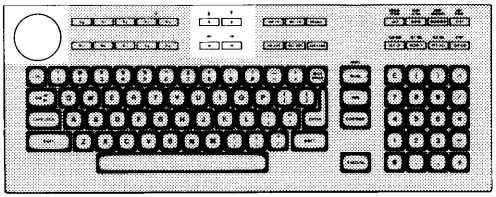
The numeric keypad provides a convenient way to enter numbers and perform arithmetic operations. Simply type in the arithmetic expression you want to evaluate, then press [EXECUTE]. The result is displayed in the lower-left corner of the screen.

Type in the following problem using the numeric keypad:

```
(26+14)/4
```

Now press [EXECUTE] to perform the calculation. The answer, `10`, is displayed in the lower-left corner of the screen.

Cursor-Control Keys



The cursor-control keys move the display cursor. The [↑] and [↓] keys allow you to scroll lines in the output area up and down. Shifted, the keys allow you to "jump" to the top and bottom of the output area. The [→] and [←] keys allow you to move horizontally along a line. Shifted, they allow you to "jump" to the left and right limits of a line. The [BACK SPACE] key works just like the [←] key.

The cursor control wheel (also called the knob) allows you to rapidly scroll the print area (with [SHIFT] pressed) or move the cursor left and right (unshifted).

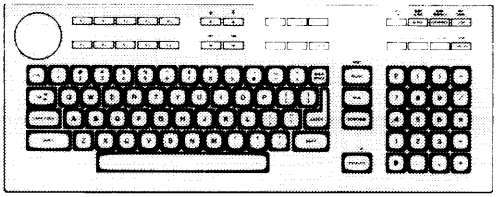
To test the horizontal movement of the cursor, type a few words and press the [←] and [→] keys. Notice that the cursor cannot be moved beyond the characters you have typed. Now rotate the wheel to move the cursor. Press [CLR LN] when finished.

To test vertical scrolling, type **EDIT** and press [EXECUTE]. Now type the following lines, pressing [ENTER] after each line (the first line may be there already, so just press [ENTER] to accept it):

```
10 !FIRST LINE
20 !SECOND LINE
30 !THIRD LINE
40 !FOURTH LINE
```

Press the [SHIFT] key and rotate the wheel to scroll the text up and down. Also try out the [↑] and [↓] keys. When you're done, press [PAUSE] to exit.

Editing Keys



The editing keys put easy character editing and line editing at your fingertips.

[EDIT]

The [EDIT] key is a typing convenience; pressing [EDIT] followed by [EXECUTE] puts the computer in program edit mode. Edit mode allows you to enter and edit program lines.

Press [EDIT], then [EXECUTE] to enter edit mode. The number 10 appears on the screen. This is a line number for a BASIC program; the computer is waiting for you to type in a line of code. If there is a program already in memory, the computer displays it on the screen. Press [PAUSE] to exit edit mode.

[RECALL]

The [RECALL] key recalls the last line that you entered, executed, or deleted. Several previous lines can be recalled this way. [RECALL] is particularly handy when you mistype a line. Instead of retyping the entire line, you can recall it, edit it using the editing keys, and enter or execute it again.

Type:

```
PRINT "1"  [EXECUTE]
```

to print the number 1 on the screen. Now press [RECALL] to recall the PRINT statement. Edit the statement to print the number 2 by positioning the cursor under the 1 and typing [2] over it. Press [EXECUTE] again. Now press [RECALL] several times to see all of the statements it remembers. Then press [CLR SCR] when finished.

[SHIFT]-[RECALL] moves forward through the recall stack.

[INS LN] [INS LN] inserts a new line above the cursor's current position (edit mode only).

Press [EDIT], then [EXECUTE]. Type in this line (if it isn't already there):

```
10 !FIRST LINE
```

Now, with the cursor somewhere on line 10, press [INS LN]. Notice that a new line number (1) is inserted before line 10. Press [PAUSE] when finished.

[DEL LN] [DEL LN] deletes the line containing the cursor (edit mode only).

Press [EDIT], then [EXECUTE]. Position the cursor to the line:

```
10 !FIRST LINE
```

and press [DEL LN]. The line is removed. To restore it, press [RECALL] to retrieve it, then [ENTER] to enter it into the program. Press [PAUSE] to exit edit mode.

[INS CHR] [INS CHR] sets insert mode, allowing you to insert characters to the left of the cursor. Press the key a second time to cancel insert mode.

Carefully type the following line exactly as shown:

```
THIS IS A TEST .
```

Position the cursor under the period and press [INS CHR]. Now type:

```
OF INSERT MODE
```

and press [INS CHR] again. The line should now look like this:

```
THIS IS A TEST OF INSERT MODE.
```

The new characters were inserted to the left of the period. Press [CLR LN] when finished.

[DEL CHR] [DEL CHR] deletes the character at the cursor's position.

Type a few words and experiment with [DEL CHR], positioning the cursor at various places on the line. Notice that if you hold the key down, characters are deleted until you release it. Delete all of the characters you typed.

[SET TAB] [SET TAB] ([SHIFT]-[RESULT]) sets a tab at the cursor's current position. Tabs remain in effect until cleared by either [CLR TAB] or the SCRATCH A statement. The SCRATCH commands are explained in *BASIC Programming Techniques*. To demonstrate [SET TAB], see [TAB].

[CLR TAB] [CLR TAB] ([SHIFT]-[PRT ALL]) clears a tab previously set at the cursor's position. To demonstrate [CLR TAB], see [TAB].

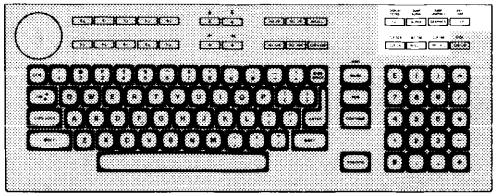
[CLR LN] [CLR LN] clears the keyboard line and message/results line.

Type a few words and press [CLR LN] to clear them.

[CLR→END] [CLR→END] clears from the current cursor position to the end of the line.

Type in a few words and use the cursor control wheel or [←] to position the cursor in the middle of the line. Press [CLR→END] to clear to the end of the line. Press [CLR LN] to clear the rest of the line.

System Control Keys



These keys control various system functions related to the display, printer, and editing operations. Most of these keys execute their functions immediately, as the key is pressed.

[EDIT]

[EDIT] types the EDIT command on the keyboard line. See the Editing Keys section for more information.

[DISPLAY FCTNS]

[DISPLAY FCTNS] ([SHIFT]-[EDIT]) sets the display-functions mode, allowing you to see special control characters (e.g., form-feed, carriage return) on the screen. Pressing this key a second time cancels the display-functions mode.

Type the following line:

```
PRINT "DISPLAY-FUNCTIONS MODE OFF"  [EXECUTE]
```

Notice the display at the top of the screen. Now press [RECALL] to recall the line, and edit it to read:

```
PRINT "DISPLAY-FUNCTIONS MODE ON"
```

Press the [DISPLAY FCTNS] key, and then press [EXECUTE]. Notice that the carriage return (CR) and line-feed (LF) control characters are now displayed. Press [DISPLAY FCTNS] again to exit display-functions mode. Press [CLR SCR] when finished.

[ALPHA]
[GRAPHICS]

[ALPHA] and [GRAPHICS] allow you to turn the alpha and graphics display modes on and off. The GRAPH binary must be loaded for these keys to function.

[DUMP ALPHA]

The [DUMP ALPHA] ([SHIFT]-[ALPHA]) key prints a complete copy of the alpha display on the default printer.

[DUMP GRAPHICS] The [DUMP GRAPHICS] ([SHIFT]-[GRAPHICS]) key prints a complete copy of the graphics display on the default printer. The GRAPH binary must be loaded for this key to function.

[STEP] [STEP] allows the programmer to step through a program, one line at a time. Using the [STEP] key to debug programs is covered in *BASIC Programming Techniques.*

[ANY CHAR] [ANY CHAR] ([SHIFT]-[STEP]) is used to find any ASCII character. First press [ANY CHAR]. Then enter a three-digit number from 000 through 255 representing the decimal equivalent of an ASCII character. The computer automatically displays the character on the screen. For a list of characters and their equivalent decimal values, see the US ASCII Character Codes table in the "Useful Tables" appendix of the *BASIC Language Reference.*

Press [ANY CHAR], then type 65 which is the decimal equivalent of "A". The "A" is now displayed in the keyboard line. Press [CLR LN] to erase it.

[CLR SCR] [CLR SCR] ([SHIFT]-[CLR LN]) clears the entire alpha screen.

Type the following BASIC command:

```
PRINT "PUT THIS MESSAGE IN THE OUTPUT AREA."
```

Now press [EXECUTE] to execute it. Press [RECALL] to recall the command and press [EXECUTE] again. Repeat this step several times to fill the screen with messages. Now press [CLR SCR] to erase all lines at once.

[RESULT] [RESULT] returns the result of the last arithmetic expression that was executed.

Press [CLR LN], then type:

```
23+45 [EXECUTE]
```

The result, `68`, is displayed in the lower-left corner of the screen. To add 123 to this value, type:

```
[RESULT] +123 [EXECUTE]
```

The new result, `191`, is now displayed. Press [CLR LN] when finished.

PRT ALL

The PRT ALL key turns the printall mode on and off, allowing keyboard operations and displayed error messages to be copied to a printall device. Press PRT ALL once to set printall "on" and again to set printall "off". The printall mode is displayed in the lower-left corner of the screen.

The screen's output area is the default printall device. Selecting an external printall device is explained in *BASIC Programming Techniques.*

Press PRT ALL to turn on printall mode. Now type in the following command:

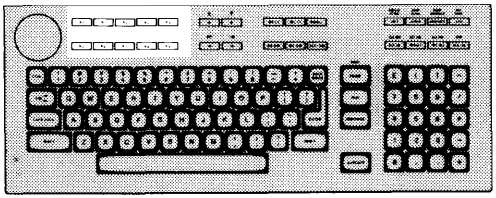
```
PRINT "THIS IS A KEYBOARD OPERATION"  EXECUTE
```

Both the PRINT command and the message itself are displayed on the screen, which is the default printall device. Now type:

```
THIS WILL CAUSE AN ERROR  EXECUTE
```

Because this is not an executable BASIC statement, an error message is displayed, both at the bottom of the screen and in the printall area at the top. This way, a log is produced of all commands typed and executed at the keyboard, along with any error messages. Press CLR SCR to clear the screen, and press PRT ALL to turn off printall mode.

Softkeys

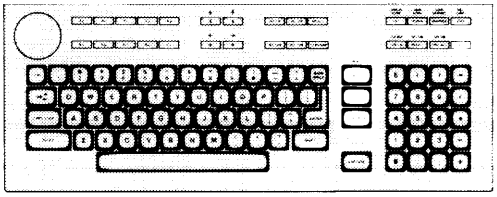


The ten keys labeled k0 through k9 are defined under program control. The program may also display a label for each defined key. Pressing a defined key tells the computer to interrupt whatever it's doing and start running another part of the program.

We call these keys "softkeys" because the program or "software" defines and labels them. Another ten softkeys (without the displayed labels) can be defined at the same time and accessed with the SHIFT key. These shifted softkeys are often referred to as **k10** through **k19**.

With KBD language extension binary loaded, softkeys are defined as typing aids.

Program Control Keys



The keys shown below allow you to control execution of the program stored in the computer's memory.

RUN	RUN starts a program running from the beginning.
PAUSE	PAUSE pauses program execution after the current line. It is also used to exit the Editor.
CONTINUE	CONTINUE resumes program execution from the point where it was paused. It is also used like ENTER or EXECUTE to respond to a program prompt.
STOP	STOP (SHIFT-CLR I/O) stops program execution after the current line. Unlike PAUSE, you cannot resume execution of a program stopped with STOP by pressing CONTINUE. To restart the program, use the RUN key.
RESET	RESET (SHIFT-PAUSE) stops program execution immediately without erasing the program from memory. The **BASIC Reset** message indicates the computer is ready for your command.
CLR I/O	CLR I/O pauses program execution when the computer is performing or trying to perform an I/O operation. Press CLR I/O instead of PAUSE when the computer is hung up on an I/O operation, since PAUSE works only after the computer finishes the current program line. Pressing CLR I/O cancels the I/O operation and pauses the program at the current line.

Task/Topic	Where to Go for Details
Character entry keys	9-2
Cursor-control keys	9-4
Editing keys	9-5
System control keys	9-8
Softkeys	9-10

HP 98203A Keyboards

9

If you have an ITF or an HP 98203B/C keyboard, ignore this chapter and refer to one of the two preceding chapters.

The keys on the HP 98203A keyboard are arranged into the following functional groups:

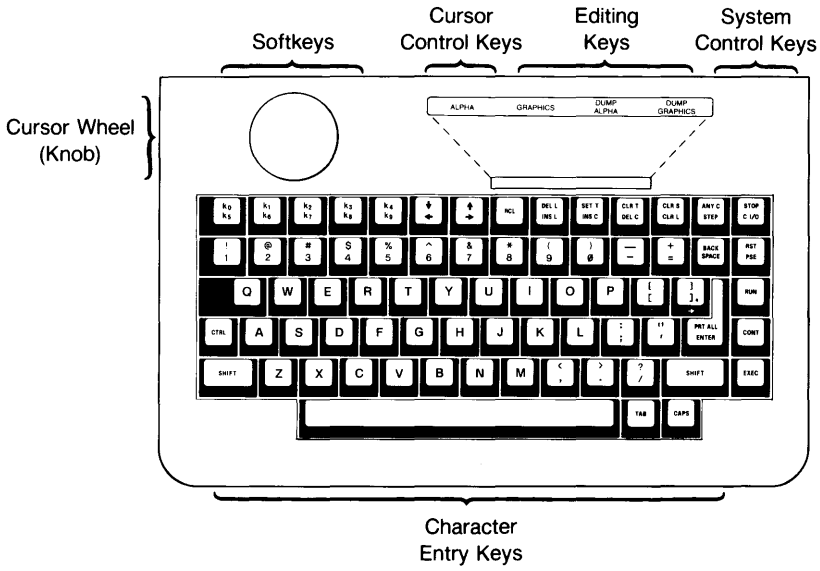


Figure 9-1. HP 98203A Keyboard

This chapter provides a handy reference guide to BASIC's key definitions for the HP 98203A keyboard. Keep in mind that other system programs may define the keys differently. Each key will be demonstrated where possible. One point to clarify: the **cursor** that we refer to in the following paragraphs is the blinking-underline that points to a location on the screen.

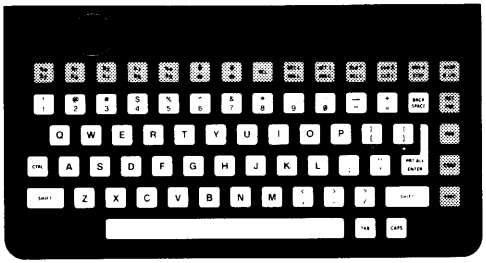
Note

Before you proceed, type:

```
SCRATCH [EXEC]
```

This clears the computer of any programs that might be left in memory from previous demonstrations.

Character Entry Keys



The character entry keys are arranged like a typewriter, but have some added features.

[CAPS]

The [CAPS] key sets the unshifted keyboard to either upper-case (which is the default after BASIC is booted) or lower-case (normal typewriter operation). The computer displays which mode the computer is in when you press the [CAPS] key.

Type a few words, then press [CAPS] and continue typing. Notice the case change. Press [CLR L] when finished.

[SHIFT]

You can enter standard upper-case and lower-case letters, using the [SHIFT] key to access the alternate case.

Type a few words, pressing [SHIFT] to change the case of the first letter of each word. Now press [CAPS] and continue typing. Notice that the alternate case accessed by [SHIFT] depends on the setting of [CAPS]. Press [CLR L] when finished.

[ENTER] The [ENTER] key has several functions:

- When a running program prompts you for data, you respond by typing the requested data and then pressing [ENTER]. This signals the program that you have provided the data and that it can resume execution. The [EXEC] key can also be used for this function.
- When typing in lines of a program, the [ENTER] key is used to store each line of program code. The [EXEC] key can also be used for this function.
- Like the [EXEC] key, the [ENTER] key can be used to execute commands and calculations.

Type `EDIT` and press [ENTER]. Notice the number `10` now displayed on the screen—this is the line number of the first line of a BASIC program. The computer is waiting for you to type in the line. Type:

```
!FIRST LINE
```

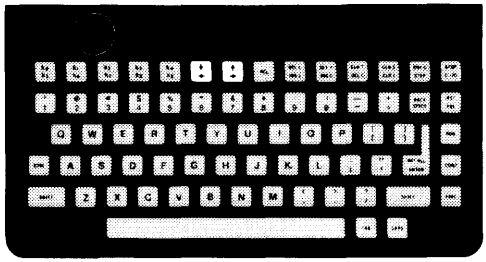
and press [ENTER]. Notice that the computer accepts the statement as a program line and displays `20` in preparation for the next one. Press [PSE] to exit.

[TAB] The [TAB] key moves the cursor forward to preset tabs. Pressing [SHIFT]-[TAB] moves the cursor back to preset tabs.

Before [TAB] can be used, a tab must be set. Press the space bar to move the cursor forward a few spaces and press [SET T]. Move the cursor back several spaces using [←], then press [TAB]. Move the cursor forward several more spaces with the space bar, then press [SHIFT]-[TAB]. To clear the tab, move the cursor to the unwanted tab position and press [CLR T].

[CTRL] The [CTRL] (control) key works like [SHIFT] to access a set of standard control characters, such as line-feed and form-feed. These characters are useful to the programmer for controlling some devices and for communicating with other computers. You probably won't need them when running programs. The available control characters are listed in the "Useful Tables" appendix of *BASIC Language Reference.*

Cursor-Control Keys



The cursor-control keys move the display cursor. The [↑] and [↓] keys allow you to scroll lines in the output area up and down. The [→] and [←] keys allow you to move horizontally along a line. The [BACK SPACE] key works just like the [←] key.

The cursor control wheel (also called the knob) allows you to rapidly scroll the output area up and down or move the cursor left and right, depending on the [SHIFT] key. With the [SHIFT] key pressed, the knob scrolls the output area up and down. Without the [SHIFT] key pressed, the knob moves the cursor left and right.

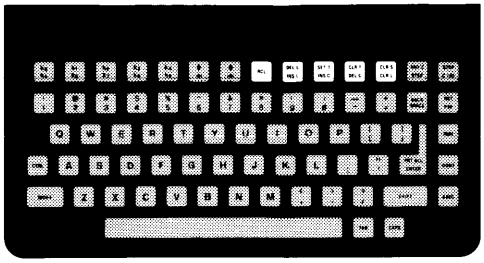
To test the horizontal movement of the cursor, type a few words and press the [←] and [→] keys. Notice that the cursor cannot be moved beyond the characters you have typed. Now rotate the wheel to move the cursor. Press [CLR L] when finished.

To test vertical scrolling, type `EDIT` and press [EXEC]. Now type the following lines, pressing [ENTER] after each line (the first line may be there already, so just press [ENTER] to accept it):

```
10 !FIRST LINE
20 !SECOND LINE
30 !THIRD LINE
40 !FOURTH LINE
```

Now, press [SHIFT] and rotate the knob to scroll the text up and down. Also try out the [↑] and [↓] keys. When you're done, press [PSE] to exit.

Editing Keys



The editing keys put easy character editing and line editing at your fingertips. Some of these keys only work when you are in edit mode, which is entered by typing:

```
EDIT [EXEC]
```

Edit mode is described in detail in *BASIC Programming Techniques.* To exit edit mode, press [PSE].

[RCL]

The [RCL] key recalls the last line that you entered, executed, or deleted. Several previous lines can be recalled this way. [RCL] is particularly handy to use when you mistype a line. Instead of retyping the entire line, you can recall it, edit it using the editing keys, and enter or execute it again.

Type:

```
PRINT "1"  [EXEC]
```

to print the number 1 on the screen. Now press [RCL] to recall the PRINT statement. Edit the statement to print the number 2 by positioning the cursor under the 1 and typing [2] over it. Press [EXEC] again. Now press [RCL] several times to see all of the statements it remembers from the last entered to the earliest entered. Then press [SHIFT]-[RCL] several times to review the statements from the earliest to the last. Press [CLR S] to exit.

[INS L] [INS L] inserts a new line above the cursor's current position (edit mode only).

Type `EDIT` [EXEC]. Then, type in this line (if it isn't already there):

```
10 !FIRST LINE
```

Now, with the cursor somewhere on line 10, press [INS L]. Notice that a new line number (1) is inserted before line 10. Press [PSE] to exit.

[DEL L] [DEL L] ([SHIFT]-[INS L]) deletes the line containing the cursor (edit mode only).

Type `EDIT` [EXEC]. Position the cursor to the line:

```
10 !FIRST LINE
```

and press [DEL L]. The line is removed. To restore it, press [RCL] to retrieve it, then [ENTER] to enter it into the program. Press [PSE] to exit edit mode.

[INS C] [INS C] sets insert mode, allowing you to insert characters to the left of the cursor. Press the key a second time to cancel insert mode.

Carefully type the following line exactly as shown:

```
THIS IS A TEST .
```

Position the cursor under the period and press [INS C]. Now type:

```
OF INSERT MODE
```

and press [INS C] again. The line should now look like this:

```
THIS IS A TEST OF INSERT MODE.
```

The new characters were inserted to the left of the period. Press [CLR L] when finished.

[SET T] [SET T] ([SHIFT]-[INS C]) sets a tab at the cursor's current position. Tabs are in effect for the keyboard line until cleared by either [CLR T] or the SCRATCH A statement. The SCRATCH commands are explained in *BASIC Programming Techniques.* To demonstrate [SET T], see [TAB].

[CLR T] [CLR T] ([SHIFT]-[DEL C]) clears a tab previously set at the cursor's position. To demonstrate [CLR T], see [TAB].

[DEL C] [DEL C] deletes the character at the cursor's position. Type a few words and experiment with [DEL C], positioning the cursor at various places on the line. Notice that if you hold the key down, characters are deleted until you release it. Delete all of the characters you typed.

[CLR L] [CLR L] clears the keyboard line and message/results line. Type a few words and press [CLR L] to clear them.

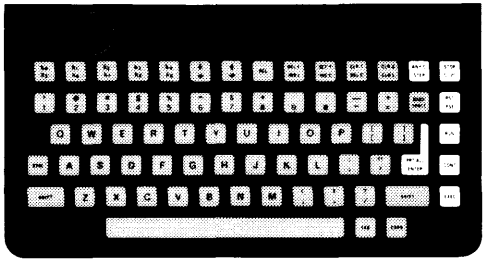
[CLR S] [CLR S] ([SHIFT]-[CLR L]) clears the entire alpha screen.

Type the following BASIC command:

```
PRINT "PUT THIS MESSAGE IN THE OUTPUT AREA."
```

Now press [EXEC] to execute it. Press [RCL] to recall the command and press [EXEC] again. Repeat this step several times to fill the screen with messages. Now press [CLR S] to erase all lines at once.

System Control Keys



The keys on the right-hand side of the keyboard control various system functions related to the display, printer and editing operations. Most of these keys execute their functions immediately, as the key is pressed.

[STEP] — [STEP] allows you to step through a program, one line at a time. Using the [STEP] key to debug programs is covered in *BASIC Programming Techniques.*

[ANY C] — [ANY C] ([SHIFT]-[STEP]) is used to find any ASCII character. First press [ANY C]. Then enter a three-digit number from 000 through 255 representing the decimal equivalent of an ASCII character. The computer automatically displays the character on the screen. For a list of characters and their equivalent decimal values, see the US ASCII Character Codes table in the "Useful Tables" appendix of the *BASIC Language Reference.*

Press [ANY C], then type 65 which is the decimal equivalent of "A". The "A" is now displayed in the keyboard line. Press [CLR L] to erase it.

[RST] — [RST] ([SHIFT]-[PSE]) stops or resets program execution immediately without erasing the program from memory. The **BASIC Reset** message indicates the computer is ready for your command.

[PRT ALL] — [PRT ALL] ([SHIFT]-[ENTER]) key turns the printall mode on and off, allowing keyboard operations and displayed error messages to be copied to a printall device. Press [PRT ALL] once to set printall "on" and again to set printall "off". The printall mode is displayed in the lower-left corner of the screen.

The screen's output area is the default printall device. Selecting an external printall device is explained in *BASIC Programming Techniques.*

Press [PRT ALL] to turn on printall mode. Now type in the following command:

```
PRINT "THIS IS A KEYBOARD OPERATION"  [EXEC]
```

Both the PRINT command and the message itself are displayed on the screen, which is the default printall device. Now type:

```
THIS WILL CAUSE AN ERROR  [EXEC]
```

Because this is not an executable BASIC statement, an error message is displayed, both at the bottom of the screen and in the printall area at the top. This way, a log is produced of all commands typed and executed at the keyboard, along with any error messages. Press [CLR S] to clear the screen, and press [PRT ALL] to turn off printall mode.

[RUN] [RUN] starts a program running from the beginning.

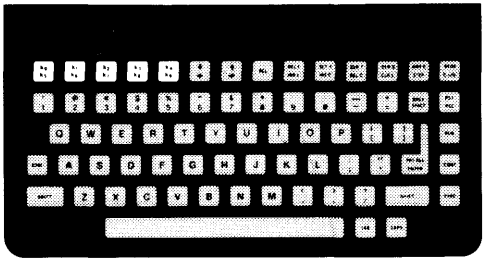
[PSE] [PSE] pauses program execution after the current line. When in edit mode, [PSE] causes the computer to exit edit mode. Some BASIC keyboard commands cannot be executed while a program is running. In this situation, you can press [PSE] to suspend program execution, type and execute your keyboard command, then resume the program with the [CONT] key (described next). (There are some keyboard commands which will not allow a program to be resumed.)

[CONT] [CONT] resumes program execution from the point where it was paused. It is also used like [ENTER] or [EXEC] to respond to a program prompt.

[C I/O] [C I/O] pauses program execution when the computer is doing an I/O operation. Press [C I/O] instead of [PSE] when the computer is hung up on an I/O operation, since [PSE] works only after the computer finishes the current program line. Pressing [C I/O] cancels the I/O operation and pauses the program at the current line.

[STOP] [STOP] ([SHIFT]-[C I/O]) stops program execution after the current line. Unlike [PSE], you cannot resume execution of a program stopped with [STOP] by pressing [CONT]. To restart the program from the beginning, use the [RUN] key.

Softkeys



The ten keys labeled [k0] through [k4] (using the [SHIFT] key) and [k5] through [k9] are defined under program control. The program may also display a label for each defined key. Pressing a defined key tells the computer to interrupt whatever it's doing and start running the designated part of the program.

We call these keys "softkeys" because the program or "software" defines and labels them.

With the KDB language extension binary loaded, softkeys can be used as typing aids.

Task/Topic	Where to Go for Details
98616A BASIC Language System Software	10-1
Disc Media Options	10-1
Disc Labels and Part Numbers	10-2
Disc Contents	10-3
BASIC 5.1 System Disc	10-3
BASIC 5.1 Language Extensions Disc	10-4
BASIC 5.1 Drivers Disc	10-5
BASIC Utilities 1 Disc	10-6
BASIC Utilities 2 Disc	10-7
BASIC HFS Utilities Disc	10-8
BASIC Manual Examples Disc	10-9

BASIC 5.1 Disc Contents 10

This chapter discusses the contents and part numbers of the discs that come with your system.

HP 98616A BASIC Language System Software

The following tables list the media options and corresponding discs supplied with the BASIC 5.1 system. Subsequent sections list and describe the files on each disc.

Disc Media Options

There are three media (disc) options available with the BASIC 5.1 system:

- Option 045: Double-sided, 3½-inch flexible discs
- Option 044: Single-sided, 3½-inch flexible discs
- Option 042: Single-sided, 5¼-inch flexible discs

Disc Labels and Part Numbers

Here are the disc labels and part numbers for each media option.

Single-Sided Discs	Double-Sided Discs
BASIC 5.1 System Disc (98616-10p00[1])	BASIC 5.1 System Disc (98616-10500)
BASIC 5.1 Language Extensions (98616-10p01[1]) BASIC 5.1 Drivers Disc (98616-10p02[1])	BASIC 5.1 Drivers and Language Extensions (98616-10501)
BASIC 5.1 Utilities Disc 1 (98616-10p05[1]) BASIC 5.1 Utilities Disc 2 (98616-10p04[1])	BASIC 5.1 Utilities Disc (98616-10502)
BASIC 5.1 HFS Utilities (98616-10p05)	BASIC 5.1 HFS Utilities (98616-10503)
BASIC 5.1 Manual Examples Disc (98616-10p06[1])	BASIC 5.1 Manual Examples Disc (98616-10504)

[1] The "p" in this suffix changes according to media option: a "2" indicates 5 1/4-inch single-sided discs; a "4" indicates 3 1/2-inch single-sided discs. For instance, the 5 1/4-inch single-sided "System" disc is 98616-10202, while the 3 1/2-inch single-sided "System" disc has part number 98616-10402.

Disc Contents

The following disc file lists show the *single-sided* disc contents, not double-sided disc contents. You should not have any problems, however, since the double-sided discs have labels that reflect the labels of both single-sided discs. For instance, the contents of the single-sided "Utilities 1" and "Utilities 2" discs are on the double-sided "Utilities" disc.

Each disc contains a file called REVID. This file has information such as copyright and revision number in it.

BASIC 5.1 System Disc

File Name	Description
SYSTEM_BA5	the "core" BASIC system; a bootable file containing a minimum language system
AUTOST	a program run automatically at boot time that loads BASIC binaries
REVID	a file with copyright and revision information

BASIC 5.1 Language Extensions Disc

File Name	Description
CLOCK	a binary that provides increased time and date capability
COMPLEX	a binary that adds complex math functions, hyperbolic functions, and the complex data type
CRTX	a binary that provides several CRT display "eXtended" capabilities
EDIT	a binary that provides an editing environment for BASIC programs
ERR	a binary that extends BASIC error messages to include an English explanation of the error
GRAPH	a binary that provides graphics capability
GRAPHX	a binary that extends "GRAPH" by providing graphics input, color plotting, and area filling capabilities
IO	a binary that provides increased Input/Output (I/O) capability
KBD	a binary that provides increased keyboard and softkey capability and HP-HIL access
KNB2_0	a BIN file that causes KNOBX to function as it does in BASIC 2.0/2.1. Only load KNB2_0 if you want KNOBX to function in 2.0/2.1 mode. Refer to the Knob section in "Porting to 3.0", *BASIC Programming Techniques* for more information
LEX	a binary that provides lexical order capability and non-U.S. keyboard support
MAT	a binary that provides increased array and matrix capabilities
MS	a binary that provides increased mass storage capability
PDEV	a binary that provides increased program development capability
TRANS	a binary that provides background Input/Output transfer capability and the use of BUFFERs
XREF	a binary that provides a cross reference capability
REVID	a file with copyright and revision information

BASIC 5.1 Drivers Disc

File Name	Description
BCD	a binary that provides the interface driver for the HP 98623 Binary Coded Decimal interface
BUBBLE	a binary that provides the device driver for HP 98259 BUBBLE Memory card
CONFIGURE	a program that helps you configure the system
CRTA	a binary that provides the device driver for non-bit-mapped displays. This binary is also included in the SYSTEM_BA5 file
CRTB	a binary that provides the device driver for bit-mapped displays. This binary is also included in the SYSTEM_BA5 file
CS80	a binary that provides the device driver for CS/80 and SS/80 type discs
DCOMM	a binary that provides the interface driver for the HP 98628 Datacomm and HP 98629 SRM interfaces
DISC	a binary that provides the device driver for non-CS/80 external disc drives
EMULATIONS	an interactive program that creates an autostart program that enables you to specify compatibility options
EPROM	a binary that provides the device driver for HP 98255 Erasable Programmable Read Only Memory cards
FHPIB	a binary that provides the device driver for HP 98625 High-speed disc interface
GPIO	a binary that provides the interface driver for HP 98622 General Purpose Input/Output interface
HPIB	a binary that provides the interface driver for the internal HPIB or the HP 98624 HP-IB interface
HFS	a binary that provides a Hierarchical File System and a few statements for managing files
HP9885	a binary that provides the device driver for HP 9885 flexible disc drives
SERIAL	a binary that provides the interface driver for the HP 98626 RS-232 Serial Interface
SRM	a binary that provides a driver and statements for SRM (Shared Resource Manager) systems
REVID	a file with copyright and revision information
RENAME_BT	utility to rename or purge the directory entries in the LIF boot boot area of an HFS disc

BASIC Utilities 1 Disc

File Name	Description
PHYREC	a set of CSUBs that allows bit-by-bit copies between a real array and a mass storage media
DUMP	a program that dumps a specific record from a disc
MASS_STOR	a program that provides several mass storage functions
CAT	a subprogram that reads the disc directory
INFO	a subprogram that returns directory information for a specific file
CREATE	a subprogram that creates a file at a specific location on the disc
INITIALIZE	a program that initializes a disc with a specific volume label and directory size (useful for modifying the default directory size of hard discs, for example); LIF format only
VERIFY_LIF	a program that determines if a volume meets HP LIF standards
CBACKUP	a program that backs-up the contents of a flexible disc that meets HP LIF standards
FBACKUP	a program that backs-up selected files of a flexible disc that meets HP LIF standards
TAPEBACKUP	a program that backs-up a CS/80 disc onto a DC600 or DC150 tape cartridge or recovers data from a tape cartridge to a CS/80 disc
MEM_UTILS	provides 5 memory resident, softkey-accessed utility subprograms for file management and editing
INTERFACES	a program that lists the interface cards currently installed in the computer, and shows whether the corresponding BASIC driver is currently loaded
HFSDISC	a subroutine loaded by several utilities to detect HFS media
VERIFY	a program that allows you to label peripheral devices (disc drives, printers, plotters, etc.), verify their proper operation, and sometimes to see example BASIC statements used to access them
REVID	a file with copyright and revision information

BASIC Utilities 2 Disc

File Name	Description
LISTER	a program that lists the contents of ASCII files
82905DUMP	a subprogram that dumps a graphics display to an HP 82905B printer
LEX_AID	a program that aids in creating lexical order tables you define
GERMAN	a file that contains default lexical order tables for German
FRENCH	a file that contains default lexical order tables for French
SPANISH	a file that contains default lexical order tables for Spanish
SWEDISH	a file that contains default lexical order tables for Swedish
FILE_STAT	a program that displays the contents of the status registers of an ASCII or BDAT file
HPIB_STAT	a program that displays the contents of the HP-IB interface's status registers
RS232_STAT	a program that displays the contents of the RS232 interface's status registers
GPIO_STAT	a program that displays the contents of the GPIO interface's status registers
SYSTEM_LD	a SYSTM file that contains the LOADER system
CONFIGER	a program that creates the CONFIG file for the Loader
CONFIG_CHK	a program that lists the CONFIG file
GDUMP_R	a CSUB that performs a raster "graphics dump" to a printer (rotated 90°, but not expanded)
GDUMP_C	a CSUB that performs a color raster "graphics dump" to a PaintJet printer
BPLOT	a file containing CSUBs (Bstore and Bload) that store and load rectangular blocks of raster data in an integer array
BACKUP	a program that allows you to make back-up copies of files and/or volumes. It is useful because it allows wildcards in file names, and can perform "incremental" back-ups (for instance, you can back-up all files modified since a specific date)
VME_DRIVER	a CSUB library that provides access to the HP 98646A VME interface
VME_TEST	a program that uses VME_DRIVER to perform a simple test of the HP 98646A VME interface
REVID	a file with copyright and revision information

BASIC HFS Utilities Disc

File Name	Description
DISC_UTIL	a collection of utilities: show on-line mass storage devices, initialize a disc, store BASIC system and binaries, check consistency of an HFS volume, and call the BACKUP/RESTORE utility
REVID	a file with copyright and revision information

BASIC Manual Examples Disc

File Name	Description
Activate	demonstrates how to turn the function box status light on and off with the HIL SEND command
Animation	demonstrates color map animation
BACKGROUND	demonstrates color map definition, non-dominant drawing, three-dimensional transformations, and knob interaction
BAR_KNOB	demonstrates the use of the knob to control dynamic displays
Bar_code	provides a program that allows you to read bar codes
Button_box	provides an example for using the buttons on a function box to cause interrupts that are trapped and to set processes in motion
CDials	demonstrates the use of "Control Dials" (9-knob) box
CIRCLES	shows that the color map can be defined to simulate an additive color scheme, a subtractive color scheme, or any arbitrary color scheme
CScale	demonstrates how to generate notes in the C musical scale using the built-in tone generator
CharCell	shows the relationship between the actual character size and the character cell size
Contour	a demonstration of a random contour map
Csize	demonstrates how to use the CSIZE statement to change the size of the character cells into which labelled characters are placed
DITHER_PAL	a program that creates a palette for dithering (for color monitors)
DumpGraph	takes an image from the frame buffer of a monochromatic CRT and sends it to an HP 82905A printer
Gdu	a subprogram that allows graphic display units instead of user defined units
Gray_Map	accepts a two-dimensional array and plots a gray map from it
Gstore	demonstrates the use of GSTORE and GLOAD in quickly replotting the unchanging part of an otherwise dynamic image
HIL_ID	provides a means for determining which HP-HIL devices are on your HP-HIL link and gives information about them
ID_MODULE	a program that decodes an ID Module's contents

BASIC Manual Examples Disc (Continued)

File Name	Description
InputSong	shows a simple "music editor" program that defines keys on the keyboard to produce musical notes
Iplot	uses incremental plotting to create characters for plotting labels in a user-defined character set
KBD_ICONS	a pictoral menu program that supports touchscreen and HP-HIL tablets, mice, etc.
Label	defines several system variables (in CSIZE, LDIR, etc.) and labels text accordingly
Ldir	demonstrates how the LDIR statement allows labelling of text on the graphics screen at any desired angle
Lem1	a demonstration of a lunar lander drawing
Lem2D	demonstrates the four basic two-dimensional graphics transformations: translation, rotation, scaling and shearing
Lorg	demonstrates how the LORG statement allows centering or cornering of labels in both the X and Y directions
MARQUEE	uses color-map animation to create a movie marquee announcing the coming attractions
Mul_press	provides an example for mapping multiple key presses from a function box
Multi_dev	demonstrates how end-of-line interrupts can be handled when they come from more than one HP-HIL device
NEW_MODELS	provides a physical model to relate parameters of the HSL model (Hue, Saturation, and Luminosity)
OdeToJoy	an example data file containing musical notes in the song "Ode to Joy", that can be read and played with the InputSong program (also on this disc)
Pause	a subprogram to pause and continue graphics output
Pen	demonstrates drawing modes on monochromatic CRTs
Pie_Chart	accepts pie chart data up to fourteen segments, each with its own label, plus title and subtitle
RIPPLES	color map animation with concentric circles

BASIC Manual Examples (Continued)

File Name	Description
Rplot	uses RPLOT statement to move subpictures, PIVOT to rotate them, and AREA INTENSITY to define shading
SET_COLOR	a program to calculate the correct cursor and text color when you set a background color
STEREO	uses non-dominant drawing and three-dimensional transformations to display red-blue stereo images which can be viewed through bi-colored glasses
STORM	demonstrates the use and speed of color map animation. A little house on the prairie is besieged by a thunderstorm
Scale	scales a matrix to fit a new minimum and maximum
Scenery	uses POLYGONS, POLYLINES, RPLOTS, and area fills to create an idyllic scene of rustic simplicity
Show	simulates the system command SHOW
SinAxes	part of the "Progressive Example" in *Graphics Techniques*. Axes are added, along with labels at appropriate points along them
SinGrdAxes	demonstrates grids and axes; used in *Graphics Techniques* to display grids and axes in the same program
SinLabel	part of the "Progressive Example" in *Graphics Techniques*. Labels are plotted after having used CSIZE, LORG and LDIR
SinLabel2	similar to SinLabel except that it draws a "bold" label
SinViewprt	part of the "Progressive Example" in *Graphics Techniques*. A viewport is defined using GDU measurements of the screen
SoundArray	generates sounds (melody and percussion) by reading data into an array and sending the values to a SOUND statement
SoundInstr	lets you experiment with SOUND statement parameters
Surface	draws a surface represented by a two-dimensional array
Symbol	demonstrates how to define and label user-defined characters with the SYMBOL statement
Touch_plot	demonstrates how the Touchscreen can be used to locate points on your screen
FONT_ED	lets you create, edit, and store custom character fonts (usable on machines with "bit-mapped alpha" displays)
REVID	a file with copyright and revision information

Notes

Index

a

ASCII file 5-9, 5-31
Available memory, determining 2-5

b

[Backspace] key 6-4
Bad sector scan 3-17
BASIC Keyboard overlays 7-2
BDAT file 5-9, 5-22, 5-31
BIN file 5-22
Binaries, determining current 2-15
Blank disc 3-16
Booting problems 1-3
[Break] key 2-12

c

Calculations, keyboard 2-4
CAT 5-18
CAT statement 5-1, 5-2, 5-10
Cataloging Individual PROG Files 5-19
Cataloging Selected Files 5-18
Cataloging to a String Array 5-19
Catalogs to External Printers, Sending 5-2
CHANGE command 6-18, 6-24
Character entry keys 7-2, 8-2, 9-2
CHECKREAD OFF statement 5-40
CHECKREAD ON statement 5-40
Choose a system 1-3
[C I/O] key 2-12
Clear line 2-18
Clearing a program 2-14
Clearing the Computer 2-23
[Clear line] key 6-8
[CLEAR TAB] key 6-8

Clock, Checking and Setting the System 2-5
Clock compatibility with HP-UX 2-5
Closed files and hierarchical directories 5-21
[CLR→END] key 6-8
[CLR IO] key 2-12
[CLR LN] key 2-18, 6-8
Commands 2-4, 6-1, 6-2
Commands vs. Program Lines, Statements as 6-2
Commenting programs 6-30
Comments 6-27
Compatibility, files 3-14
Computer 1-2
Computer, Clearing the 2-23
[CONT] key 2-12
[CONTINUE] key 2-12
Contrast Between Documented and Undocumented Programs 6-28
COPY statement 5-32
Copying a file 5-32
Copying an entire volume 5-32
Copying Files and Volumes 5-32
Copying HFS Files 5-34
Copying LIF Files 5-33
Copying lines (by changing line numbers) 6-12
Copying Program Segments 6-25
Copying SRM Files 5-34
COPYLINES command 6-12, 6-18, 6-25
Correcting Typing Mistakes 6-4
COUNT statement 5-18
CREATE DIR statement 5-9, 5-11
CREATE statement 5-9, 6-35
Creating vi-compatible files 6-35
Criteria for choosing directory format 3-14
CRTA binary 2-16
CRTB binary 2-16
Current binaries, determining 2-15
Current line 6-7
Current system devices, determining 2-15
Current working directory 5-4, 5-21
Current Working Directory, Changing the 5-6
Cursor-Control keys 7-5, 8-5, 9-4

d

DEF FN statement 4-7
Default drive 2-15
Default Volume 5-1, 5-2, 5-5
Default Volume, Changing the 5-6, 5-7
Default Volume, Specifying Files on the 5-2
DEL command 6-18, 6-26
[DEL CHR] key 6-8
[Delete char] key 6-8
[Delete line] key 6-11
Deleting and Recalling Lines 6-11
Deleting Multiple Lines 6-26
[DEL LN] key 6-11
Device selector 5-2
Device selector (in volume specifier) 3-5
Device type (in volume specifier) 3-5
Directories 3-16
Directories and Files, Using 5-1
Directories, hierarchical 3-10
Directories, Purging SRM 5-38
Directory 5-3
Directory Access Permission, HFS 5-24
Directory, adding 5-8
Directory, Creating Files and Other Directories within a 5-8
Directory format, choosing 3-13
Directory formats (recommendations) 3-13
directory path 3-12
Directory specifiers 5-3
Directory structure:
  HFS 5-13
  LIF 5-13
  SDF 5-13
Disc format, checking 3-9
Disc format, choosing 3-13
Disc interleave 3-19
Disc organization 3-1
Disc sector size 3-18
Disc sectors 3-18
Disc volume size (flexible) 3-18
Discs, formatting 3-16
Documented and Undocumented Programs, Contrast Between 6-28

Documenting programs 6-27
Drive, default 2-15
DUMP DEVICE IS statement 2-15
Dump device, system 2-15

e

EDIT binary 6-3
EDIT command 6-14
[EDIT] key 2-17
EDIT mode 2-7, 6-3, 6-6
EDIT mode, exiting the 6-12
EDIT Mode, More Details about Getting into 6-14
Editing keys 7-7, 8-6, 9-5
Editing Operations, Global 6-18
Editing programs 6-1
Editing the Current Line, Keys Used for 6-7
Enabling Checkread Verification 5-40
END statement 2-12, 4-7
Entering programs 6-1
Exclusive access of SRM Files 5-31
Extended character set 7-4
Extensible file 5-12
Extensible files 3-14
Extent size 5-12

f

File Access Permission, HFS 5-24
File catalogs 5-13
File compatibility 3-14
File entries in directories (LIF) 3-16
File, extent size of a 5-12
File management, general 5-21
File name 3-16, 5-3
File names 3-2
File names, listing 5-18
File protection 5-21
File size 3-16
File specifier components 5-3
File specifiers 5-3
File specifiers in mass storage statements 5-22
Files and hierarchical directories, closed 5-21

Files and hierarchical directories, open 5-21
Files and Other Directories within a Directory, Creating 5-8
Files and Volumes, Copying 5-32
Files, Cataloging Selected 5-18
Files (description) 3-2
Files, extensible 3-14
Files, Finding and Specifying 5-1
Files on the Default Volume, Specifying 5-2
Files, Purging 5-35
Files, Purging SRM 5-38
Files, Renaming 5-34
Files, Using Directories and 5-1
FIND command 6-18, 6-22
Finding Textual Patterns 6-22
FNEND statement 4-7
Format, directory (recommendations) 3-13
Format of disc, checking 3-9
Format of disc, choosing 3-13
Format options (with INITIALIZE) 3-18
Formatting discs 3-16

g

GET statement 4-1, 4-4, 4-5
GET to Specify Run Line 4-6
GET with Automatic Program Clearing 4-5
Global Editing Operations 6-18

h

HFS catalog contents:
- Access rights 5-16, 5-17
- Data and Time 5-17
- Day and time 5-15
- File name 5-15
- File type 5-15, 5-17
- Group identifier 5-16
- Number of records 5-15, 5-17
- Owner identifier 5-16
- Record length 5-17
- Record size 5-15

HFS classes of users:
GROUP 5-24
OTHER 5-24
OWNER 5-24
HFS Directories and Files, Space Allocation for SRM and 5-12
HFS directories, Purging 5-37
HFS directories, storing 5-12
HFS Directory Access Permission 5-24
HFS directory format 3-10, 3-13
HFS directory structure 5-13
HFS file 5-3
HFS File Access Permission 5-24
HFS Files, Copying 5-34
HFS Files, Non-Contiguous Storage of SRM and 5-12
HFS files, Purging 5-37
HFS files, storing 5-12
HFS hierarchical-directory volume 5-7
HFS permission bits 5-24
HFS permissions:
READ 5-25
SEARCH 5-25
WRITE 5-25
HFS volume labels 5-39
HFS volume overhead 3-15
Hieracrhical Directories, Creating and Using 5-7
Hierarchical directories 3-10
Hierarchical directories, closed files 5-21
Hierarchical directories, open files 5-21
Hierarchical directory 5-4
Hierarchical Directory Capabilities 5-11
Hierarchical organizations 3-4
Hierarchical-directory volume, HFS 5-7
Hierarchical-directory volume, SRM 5-7
Hierarchy, Example 5-7
HP-UX clock compatibility 2-5
HP-UX file 5-22
HP 46084 ID Module 6-32
HP 9133 disc drive 5-6
HP 98203A Keyboard 2-3. 9-1
HP 98203B/C Keyboard 8-1
HP 98203C Keyboard 2-3

i

ID Module, HP 46084 6-32
ID Module's Contents, Reading an 6-32
ID PROM, Reading an 6-32
INDENT command 6-18, 6-19, 6-22
Indentation Bounds 6-22
Indentation, Removing 6-22
Indenting a Program 6-19
Indicators, program-status 2-10
Initialization 3-16
INITIALIZE 3-16
INITIALIZE statement 5-39
INPUT statement 4-14
[INS CHR] key 6-8
[Insert char] key 6-8
Inserting lines 6-10
[Insert line] key 6-10
[INS LN] key 6-10
Interleave, disc 3-19
ITF Keyboard 2-3, 7-1

k

KBD binary 2-17
KEY LABELS ON statement 4-3
Keyboard calculations 2-4
Keyboard commands 2-4
Keyboard, HP 98203A 9-1
Keyboard, HP 98203B/C 8-1
Keyboard Input Line 2-5
Keyboard, ITF 7-1
Keyboard, live 4-9
Keyboard overlays, BASIC 7-2
Keyboard Uses:
  Control program execution 2-4
  Load and run programs 2-4
  Perform calculations 2-4
  Type in and execute commands 2-4, 2-5
  Type in, edit, and store programs 2-4

Keyboard:
HP 98203A 2-3
HP 98203C 2-3
ITF 2-3
Keyboards, available 2-3
Keys Used for Scrolling the Program 6-9
Keyword 6-1
Keywords, letter-case in 2-2

L

Labels, softkey 2-7
Labels, volume 3-16
Letter-case significance 2-2
Letters:
Lower-case 6-6
Upper-case 6-6
LIF catalog contents:
Address 5-14
File name 5-14
File type 5-14
Mass storage volume specifier 5-14
Number of records 5-14
Protect code 5-14
Record size 5-14
Volume name 5-14
LIF data area gap 5-35
LIF directory format 3-13
LIF directory gap 5-35
LIF directory structure 5-13
LIF file 5-3
LIF Files, Copying 5-33
LIF protect codes 5-22
LIF volume labels 5-39
LINPUT statement 4-14
LIST BIN statement 2-16
LIST KEY statement 2-21
LIST statement 2-13, 6-6, 6-16
Listing a program 6-13
Listing a Program, A Closer Look at 6-16
Live keyboard 4-9
LOAD KEY command 2-22

LOAD KEY statement 2-21, 2-22
LOAD statement 4-1, 4-4
Loading programs 4-1, 4-4
Loading typing-aid softkeys 2-22
LOCK statement 5-31
Locking and Unlocking SRM Files 5-31, 5-32
Locks, SRM Passwords and 5-27
Lower-case letters 6-6
Lower-case letters in keywords 2-2

m

Mass storage concepts 3-1
MASS STORAGE IS (MSI) statement 5-11
MASS STORAGE IS statement 5-21
Mass storage organization 3-1
Mass storage volume format (recommendations) 3-13
Mass storage volume formats, checking 3-9
Mass storage volume formats, choosing 3-13
Mass storage volume specifiers (examples) 3-7
Mass storage volumes 3-3
Memory, determining available 2-5
Menu Indicator, Softkey (ITF Keyboard Only) 2-8
[Menu] key 2-13, 4-3
MOVELINES command 6-18, 6-25
Moving Lines into a Subprogram 6-26
Moving Program Segments 6-25
msus (use "msvs") 3-5
msvs 3-5, 3-7

n

NO HEADER statement 5-18
Numeric keypad 7-6, 8-4

o

ON KEY statement 2-7
Open files and hierarchical directories 5-21
Order of devices searched by boot ROM 1-6
Overhead, HFS volumes 3-15

p

Passwords and Locks, SRM 5-27
Passwords, specifying SRM 5-31
path 3-12
[PAUSE] key 2-12
PAUSE statement 2-12
Pausing and Stopping Programs 2-12
PDEV binary 6-15
Performing keyboard calculations 2-4
Permission bits 5-25
PERMIT parameters 5-26
PERMIT statement 5-24, 5-25
power switch 1-2
power up computer 1-2
Power-on state 2-10
Prerun 4-7
Preventing program listings 6-31
PRINT LABEL statement 5-40
PRINTALL IS CRT statement 2-15
Printall printer 2-15
PRINTER IS CRT statement 2-15
Printer, system 2-15
Problems with loading 1-3, 1-5
PROG file 5-19, 5-22
PROG Files, Cataloging Individual 5-19
Program control keys 7-9, 8-12
Program currently in memory 2-13
Program Execution 4-8
Program Execution, Example of Controlling 4-9
Program line 6-2
Program lines, entering 6-4
Program Lines, Statements as Commands vs. 6-2
Program-status indicators 2-10
Programs, loading 4-1, 4-4
Programs, running 4-1, 4-7
Protect code length 5-22
Protect code, removing 5-23
Protect codes, LIF 5-22
PROTECT statement 5-11, 5-22, 5-28
Protecting files 5-21
[PSE] 2-12

PURGE on LIF Directories, Effects of 5-35
PURGE statement 5-11, 5-35, 5-37, 5-38
Purging Files 5-35
Purging HFS directories 5-37
Purging HFS files 5-37
Purging SRM Files and Directories 5-38

r

RE-SAVE statement 6-35
RE-STORE KEY command 2-21
READ LABEL statement 5-39
Readable Programs 6-27
Reading an ID Module's Contents 6-32
Reading an ID PROM 6-32
[RECALL] key 2-13, 6-11
Recalling Lines, Deleting and 6-11
REM statement 6-27, 6-30
REN command 6-18, 6-19
RENAME statement 5-11, 5-22, 5-34
Renaming Files 5-34
Renumbering a program 6-19
[RESET] key 2-13
Root directory 5-4, 5-7
RUN command 4-7
[RUN] key 4-7
Run light 2-11
Running a program 2-10, 4-1, 4-7, 6-13
Running Programs, Loading and 4-1

s

SAVE statement 4-4, 6-33, 6-34
SCRATCH A command 2-23, 5-21
SCRATCH A statement 2-8
SCRATCH BIN command 2-23
SCRATCH C command 2-23
SCRATCH command 2-14, 2-23
SCRATCH KEY command 2-23
SCRATCH R command 2-23
Scrolling the Program, Keys Used for 6-9
SDF directory structure 5-13
Search-and-Replace Operations 6-24

Sector scan (INITIALIZE) 3-17
Sector size, disc 3-18
Sectors, disc 3-18
SECURE statement 6-31
Security, Software 4-3, 6-31
SELECT statement 5-18
Selector, device (in volume specifier) 3-5
SET KEY statement 2-22
SET TIMEDATE statement 2-5
[SET TAB] key 6-8
Size of disc sectors 3-18
Size of flexible disc volumes 3-18
Softkey Labels 2-7, 2-19
Softkey Menu Changes (ITF Keyboards Only), Typing-Aid 6-15
Softkey-edit mode, exit the 2-18
Softkey-Menu Indicator (ITF Keyboard Only) 2-8
Softkeys 7-12, 8-11, 9-10
Softkeys, Typing-Aid 2-6
Software Security 4-3, 6-31
space bar 1-2
Spared sectors (INITIALIZE) 3-17
SRM access capability:
  MANAGER 5-27
  READ 5-27
  WRITE 5-27
SRM and HFS Directories and Files, Space Allocation for 5-12
SRM and HFS Files, Non-Contiguous Storage of 5-12
SRM catalog contents:
  File name 5-17
SRM directories, storing 5-12
SRM directory format 3-10
SRM file 5-3
SRM Files and Directories, Purging 5-38
SRM Files, Copying 5-34
SRM Files, Locking 5-31
SRM Files, Locking and Unlocking 5-32
SRM files, storing 5-12
SRM hierarchical-directory volume 5-7
SRM Passwords and Locks 5-27
SRM passwords, specifying 5-31
SRM Volumes 5-34

Statement 6-1
Statements as Commands vs. Program Lines 6-2
[Stop] key 2-12
STOP statement 2-12
Stopping Programs, Pausing and 2-12
Storage media 3-1
STORE KEY command 2-21
STORE KEY statement 2-21
STORE statement 4-4, 6-33, 6-34
Storing a program 6-1, 6-13, 6-33
Storing the Line 6-4
String Array, Cataloging to a 5-19
SUB statement 4-7
SUBEND statement 4-7
Subordinate directory 5-8
Syntax 6-5
System Clock, Checking and Setting the 2-5
System control keys 7-10, 8-9, 9-8
System devices, determining current 2-15
System Disc Utility 3-16
System dump device 2-15
[System] key 2-8
System Messages 2-5
System printer 2-15
System state 2-10
SYSTEM$("AVAILABLE MEMORY") function 2-5, 2-15
SYSTEM$("DUMP DEVICE IS") function 2-15
SYSTEM$("MSI") function 2-15
SYSTEM$("PRINTALL IS") function 2-15
SYSTEM$("PRINTER IS") function 2-15
SYSTEM$("SERIAL NUMBER") function 6-32
SYSTEM$("VERSION\:
BASIC") function 2-16
ERR") function 2-16

t

Time, Checking and Setting the System 2-5
Time stamps, compatibility with HP-UX 2-5
Time stamps, files 3-16
TIMEZONE IS statement 2-5
TRANSFER 3-14
TRANSFER statement 4-14
Turn on computer 1-2
Typing Mistakes, Correcting 6-4
Typing-Aid Softkeys 2-6
Typing-Aid Softkeys:
  Defining Programmatically 2-22
  Examples of Re-Defining 2-17
  Files 2-21
  Listing the Current Definitions 2-21
  Loading Definitions 2-22
  Memory Available for Definitions 2-20
  Menu Changes (ITF Keyboards Only) 6-15
  Re-Defining 2-17
  Restoring Power-Up Definitions 2-22
  Storing Definitions 2-21
  Example Definition 2-7

u

Undocumented Programs, Contrast Between Documented and 6-28
Unit number (in volume specifier) 3-5
UNLOCK statement 5-31
Unlocking SRM Files, Locking and 5-32
Upper-case letters 6-6
Upper-case letters in keywords 2-2
User 1 softkey menu 2-8
User 2 softkey menu 2-9
User 3 softkey menu 2-9
[User] key 2-8

V

vi-compatible files, creating 6-35
Volume 5-3
Volume, default 5-1
Volume format (recommendations) 3-13
Volume formats, checking 3-9
Volume formats, choosing 3-13
Volume labels 3-16
Volume labels:
  HFS 5-39
  LIF 5-39
  reading 5-39
  writing 5-40
Volume number (in volume specifier) 3-5
Volume size 3-16
Volume specifiers 3-5, 5-3
Volume specifiers (examples) 3-7
Volumes 3-3
Volumes, Copying Files and 5-32

MANUAL COMMENT CARD

Using the BASIC 5.0/5.1 System

HP Part Number 98613-90000 11/87

Please help us improve this manual. Circle the numbers in the following statement that best indicate how useful you found this manual. Then add any further comments in the spaces below. **In appreciation of your time, we will enter your name in a quarterly drawing for an HP calculator.** Thank you.

The information in this manual:

Is poorly organized	1	2	3	4	5	Is well organized
Is hard to find	1	2	3	4	5	Is easy to find
Doesn't cover enough	1	2	3	4	5	Covers everything
Has too many errors	1	2	3	4	5	Is very accurate

fold ——

Particular pages with errors? _____

_____

Comments: _____

_____

_____

_____

_____

_____

_____

Name: _____

Job Title: _____

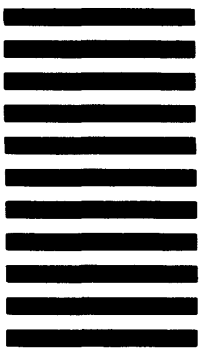
Company: _____

Address: _____

_____

☐ Check here if you wish a reply.

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES



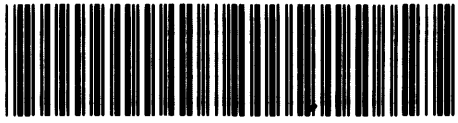
BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 37 LOVELAND, COLORADO

POSTAGE WILL BE PAID BY ADDRESSEE

Hewlett-Packard Company
Attn: Customer Documentation
3404 East Harmony Road
Fort Collins, Colorado 80525

HP Part Number
98613-90000
Microfiche No. 98613-99000
Printed in U.S.A. 11/87



98613-90630
For Internal Use Only