Using HP BASIC/UX 6.2

HEWLETT
PACKARD

HP Part No. E2040-90001
Printed in USA

Printing History

First Edition—August 1991

Contents

1 Using This Manual

This manual describes how to use HP BASIC/UX version 6.2. You can find information on:

- Entering and leaving BASIC/UX.
- Using BASIC/UX with X Windows and HP VUE.
- Performing simple BASIC/UX operations.
- Understanding mass storage concepts.
- Using directories and files.
- Editing, storing, loading and running BASIC/UX programs.
- Using HP-UX commands in BASIC/UX.
- Creating environment and autostart files.
- Using your ITF Keyboard with BASIC/UX.

For information on installing your BASIC/UX system, refer to *Installing and Maintaining HP BASIC/UX 6.2*.

BASIC/UX Compatibility

BASIC/UX is the HP-UX implementation of HP BASIC. It provides HP Series 200/300 BASIC functionality running "on top of" the HP-UX operating system. BASIC/UX is highly compatible with BASIC/WS (the workstation implementation of HP BASIC).

Note For information about the differences between BASIC/UX and BASIC/WS, refer to *HP BASIC 6.2 Porting and Globalization*, "BASIC/UX Differences and Enhancements."

A BASIC/UX system consists of several *system software* components:

- **BASIC/UX** provides the HP Series 200/300 BASIC programming language running in the HP-UX environment.
- **HP-UX** is Hewlett-Packard's implementation of the UNIX operating system. (Version 6.2 of BASIC/UX is compatible with version 8.0 of HP-UX.)
- **X Windows** provides a multi-tasking window system for bit-mapped displays in the HP-UX environment. (X Windows is part of HP-UX 8.0 and may be configured in your system at your option.)
- **HP VUE** (HP Visual User Environment) provides a user interface for the X Windows/HP-UX environment. (HP VUE is part of HP-UX 8.0 and may be configured in your system at your option.)

Notations Used in this Manual

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

If You See ...	**It Means ...**
`COMPUTER FONT`	something typed by you or the computer.
italic font	replace the italic word with your own entry. (Replace *file_name* with a file name, such as `Myfile`).
(Break)	an actual keycap on the keyboard.
`Set Tab` or `k`	a softkey label as shown on your CRT, or an inverse-video character.

All references to keycaps in the text apply to ITF keyboards.

The System Administrator

The role of the system administrator is to set up (install) and manage your system. For example, the system administrator will:

- Add disks to the system.
- Add new users to the system.
- Perform system back-ups.
- Provide information and support to users.

If *you* happen to be the system administrator, you should be familiar with the sections concerning system administration in *Installing and Maintaining HP BASIC/UX 6.2.*

For information on problems with BASIC/UX software, read the file called DEFECTS in the HP-UX directory installed with the BASIC/UX product. The directory path is `/usr/lib/rmb/newconfig`.

2

Entering and Leaving BASIC/UX

After you install your system (see *Installing and Maintaining HP BASIC/UX 6.2*), you are ready to use BASIC/UX. This chapter explains how to log in to HP-UX, how to load BASIC/UX, how to quit BASIC/UX, and how to log out of HP-UX.

Signing on to the System (login)

Before you begin, obtain the following information from your system administrator:

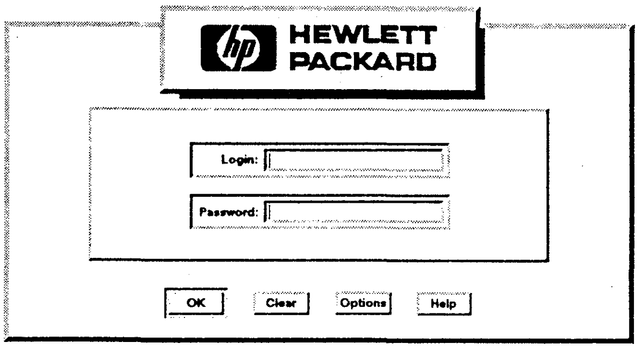
- Your user name.
- Your password.
- Your terminal type.

Logging In With HP VUE

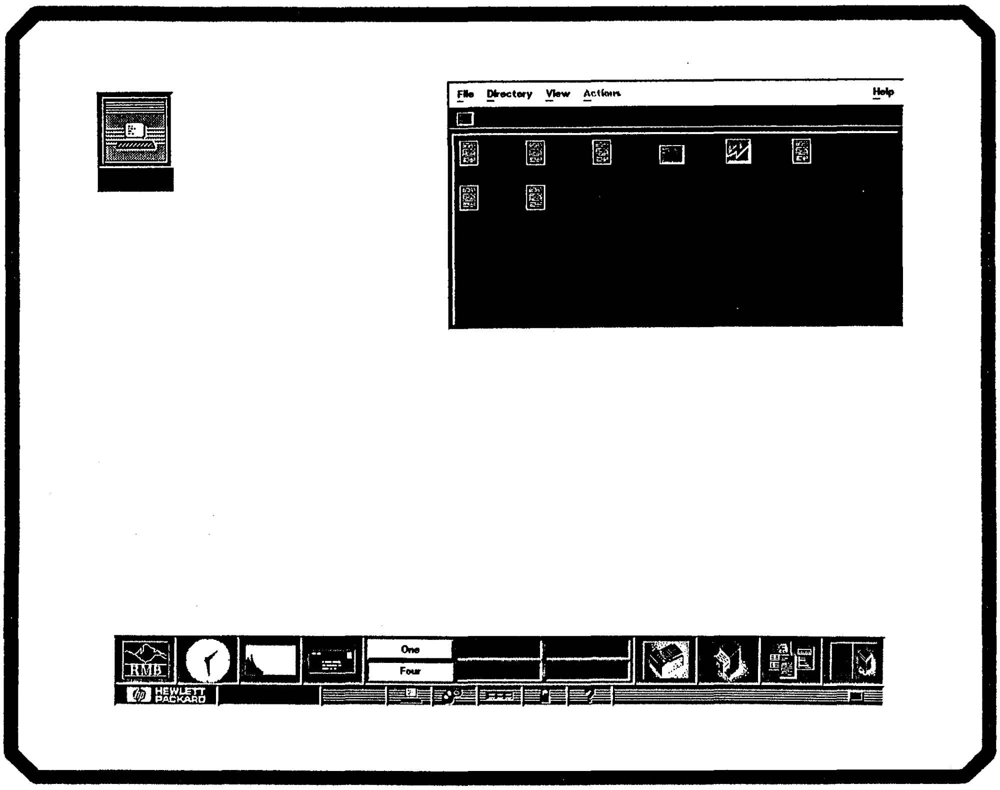
If your system is configured with HP VUE, the following screen will appear when you turn the computer on.

Note If you have purchased an HP BASIC/UX 6.2 system with pre-installed software (with BASIC/UX and HP-UX installed on the hard disk at the factory), HP VUE will be part of the pre-installed configuration.

If you are installing the software yourself, HP VUE is part of HP-UX 8.0, but you will have to configure it to run in order to use it.



Just type your login and press `Return`, then type your password and press `Return`. A screen similar to the following one will appear:



Refer to the *HP Visual User Environment User's Guide* for further information about the HP VUE user interface.

Now you can go on to "Loading BASIC/UX With HP VUE" later in this chapter.

Logging In Without HP VUE

If you are not using HP VUE, the following prompt should appear when you turn on the computer:

```
login:
```

Note If you don't see the "`login:`" prompt, press Return a few times. If you see the *system prompt* ("$" or a prompt similar to "`[xyz]:/users/bob:`"), you are already logged in.

If you see the message, `login:`, continue as follows:

1. Type your user name beside `login:`

   ```
   login: leslie
   ```

   If you make a typing mistake, press Break and try again. (The computer won't accept Back space at this point.)
2. When your user name is correct, press Return.
3. You should see `password:` on the screen. Type your password and press Return. (You won't see the password typed on the screen.)
4. At this point you may see the message `TERM = (hp)` and should type your terminal type. The terminal type tells the system how to interact with display terminals. For example, the Series 300 high resolution monitor's terminal type is 300h.

   If you do not know your terminal type you can press Return and continue. However, you may experience some display difficulties until you log in with the correct terminal type.

Once the system prompt appears, go on to "Loading BASIC/UX Without HP VUE" later in this chapter.

Problems During Login

If you see the message ...	It means ...

`Login incorrect.`
`login:`
	You made a typing mistake in your user name or password; try again. If you keep having problems, see your system administrator.

`Your password has expired. Choose a new one.`
`Changing password for` *your_user_name*
`New Password:`
	Type a new password—see the next section for how to do this.

`Maximum number of users already logged in.`	You'll have to wait until someone else logs out before you can log in.

For more information about logging in, see *A Beginner's Guide to HP-UX*.

Creating or Changing Your Password

If you need to create a password or change your password, first log in (or have your system administrator log in) to the system. After you log in, you should see a prompt similar to:

```
$
```

You are in HP-UX. You can create or change your password and start BASIC/UX by following the instructions in this section.

If your system is configured to automatically start the X Window System, you should be able to follow the instructions in this section. Just use your mouse to make your "system" or "console" window the active window, then follow the instructions below.

Note If you are using HP VUE, refer to the *HP Visual User Environment User's Guide* for information about creating or changing your password.

Creating a Password

Create your password according to the following rules:

- A password must contain at least six characters.
- At least two characters must be letters (upper- or lower-case).
- At least one character must be numeric or a special character (for example, @, -, _, or $).

Create a password that you can remember, but is not easy for others to guess. Here are some sample passwords using the above constraints.

```
number1                         super-man
$money$                         24^gold |r@t@t@\
```

Changing Your Password

Change your password periodically to protect your work. A new password must differ from the current one by at least three characters. To change your password:

- Log in to HP-UX by typing your user name and current password.
- At the system prompt, type:

  `passwd` (Return)   *Be sure* `passwd` *is typed in lower case letters.*

- Complete entries requested on the screen:

```
Changing password for leslie
Old password:                Type your old password
New password:                Type your new password
Re-enter your password:      Type the same password
```

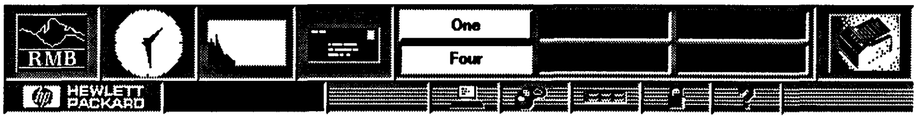
The next time you login, you must use your new password.

If you get stuck in `passwd` and you want to exit without changing anything, press (Break). (Refer to the *HP-UX Reference* manual for more information.)

Loading BASIC/UX into Memory

Loading BASIC/UX With HP VUE

If you have logged in with HP VUE, your VUE "Workspace Manager" should appear at the bottom of the screen. If you have a factory pre-installed BASIC/UX configuration, there will be a BASIC/UX ("RMB") icon at the left-hand side of the Workspace Manager, as shown below:



You can use the BASIC/UX icon in two ways:

- *As a "push button"* — just move the mouse to the icon and click the left mouse button to start BASIC. An HP Terminal window will appear, and the "rmb" command will be executed within it to start BASIC. An HP BASIC window will then appear (see the next section).
- *As a "drop zone"* — you can "drag" programs from the HP VUE File Manager and "drop" them on the icon. The HP Terminal and HP BASIC windows will appear, and the selected program will run. For further information about using the HP VUE File Manager, refer to the *HP Visual User Environment User's Guide.*

The BASIC/UX icon is part of the customized HP VUE configuration provided in the pre-installed software. This customized HP VUE configuration is also provided on the HP BASIC/UX 6.2 tape. If you are installing BASIC/UX from tape and you want to use this configuration, you will need to add a line to your `vuewmrc` file as part of the installation process. Refer to *Installing and Maintaining HP BASIC/UX 6.2* for details.

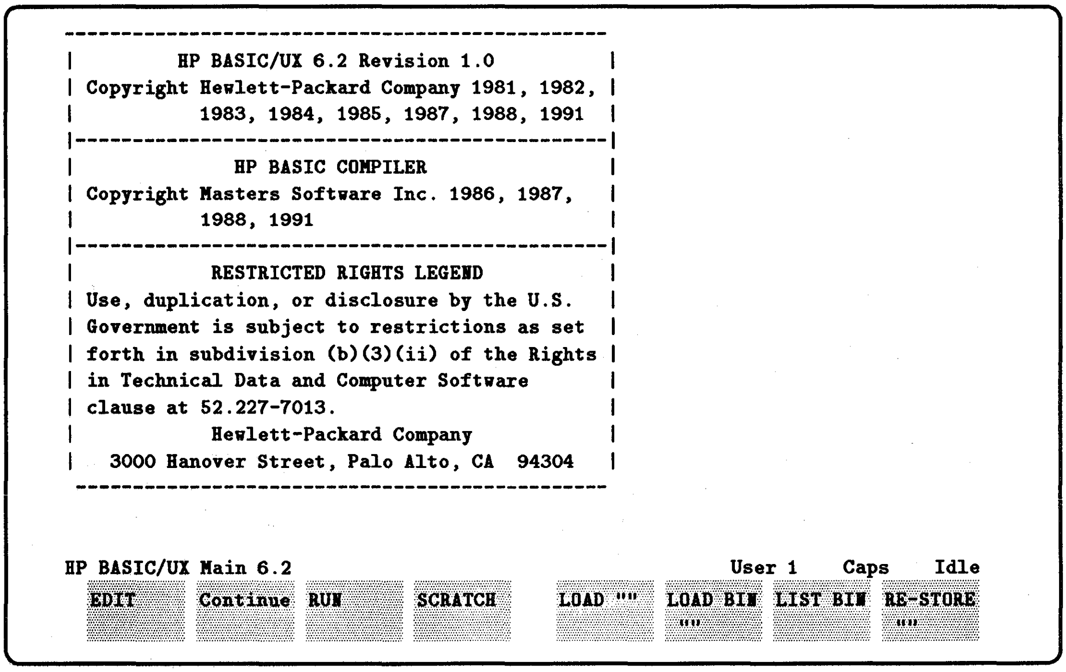
Loading BASIC/UX Without HP VUE

If BASIC/UX is not already running, you can load it by executing the following command from the system prompt. (If X Windows is running, you will have to execute the command from an active window.)

```
rmb Return
```

Note Your system administrator can configure your system so that X Windows, BASIC/UX, or both automatically load when you log in.

If X Windows is running, you'll see a new window appear. (See "Using BASIC/UX in the X Window System" for more details.) The BASIC/UX startup screen (or X Window) will look like the example below:



```
 ---------------------------------------------
|          HP BASIC/UX 6.2 Revision 1.0        |
| Copyright Hewlett-Packard Company 1981, 1982, |
|          1983, 1984, 1985, 1987, 1988, 1991  |
|----------------------------------------------|
|               HP BASIC COMPILER              |
| Copyright Masters Software Inc. 1986, 1987,  |
|          1988, 1991                          |
|----------------------------------------------|
|             RESTRICTED RIGHTS LEGEND         |
| Use, duplication, or disclosure by the U.S.  |
| Government is subject to restrictions as set |
| forth in subdivision (b)(3)(ii) of the Rights |
| in Technical Data and Computer Software      |
| clause at 52.227-7013.                       |
|             Hewlett-Packard Company          |
|   3000 Hanover Street, Palo Alto, CA  94304  |
 ---------------------------------------------

HP BASIC/UX Main 6.2                                 User 1    Caps    Idle
  EDIT     Continue  RUN      SCRATCH     LOAD ""  LOAD BIN  LIST BIN  RE-STORE
                                                      ""                  ""
```

The BASIC/UX Startup Screen

If the BASIC/UX screen or window does not appear, see your system administrator.

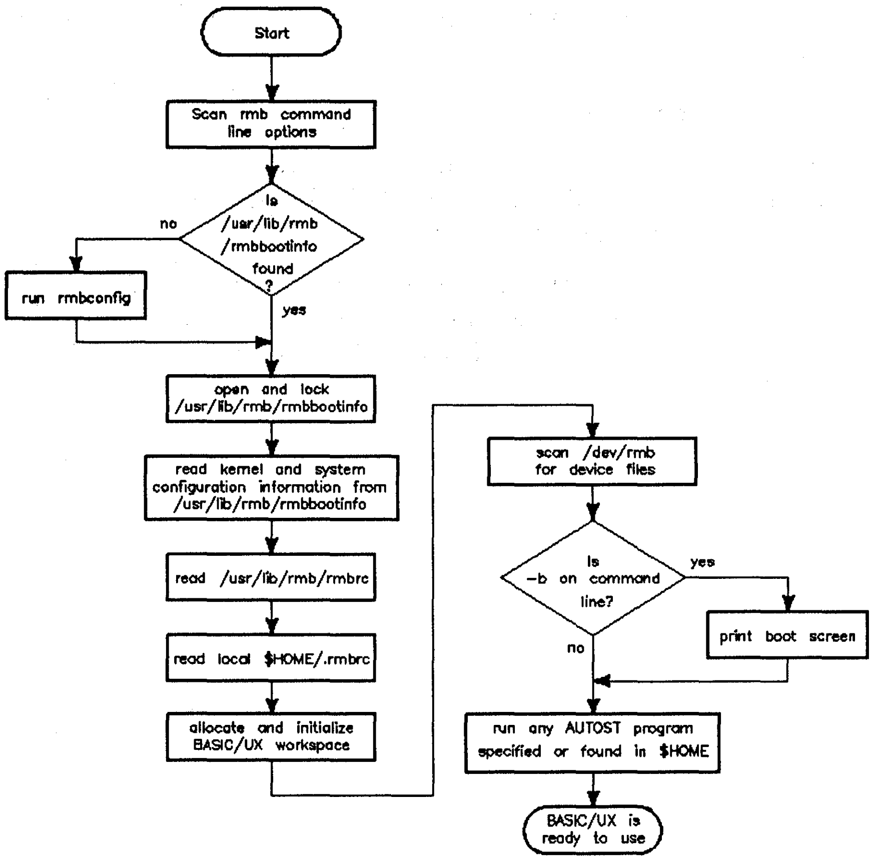
If you are in the X Window System and you see the BASIC/UX window with the message:

```
rmb: fatal internal error
```

you may have too many processes for BASIC/UX to start. Stop some processes or destroy some windows and retry BASIC/UX. If you still can't start BASIC/UX, see your system administrator.

Once you become familiar with the system, you may want to investigate the options available for **rmb**. For details on the **rmb** command, refer to appendix A, "HP-UX Command Reference," in *Installing and Maintaining HP BASIC/UX 6.2*.

The following flow chart shows the internal HP BASIC/UX boot process.


Start
Scan rmb command line options
Is /usr/lib/rmb /rmbbootinfo found ?
no
yes
run rmbconfig
open and lock /usr/lib/rmb/rmbbootinfo
read kernel and system configuration information from /usr/lib/rmb/rmbbootinfo
read /usr/lib/rmb/rmbrc
read local $HOME/.rmbrc
allocate and initialize BASIC/UX workspace
scan /dev/rmb for device files
Is -b on command line?
yes
no
print boot screen
run any AUTOST program specified or found in $HOME
BASIC/UX is ready to use

HP BASIC/UX Boot Process

Quitting BASIC/UX (QUIT)

To exit BASIC/UX, type:

```
QUIT (Return)
```

(BYE is an "alias" for QUIT.)

Logging Out of HP-UX: exit

If you logged in to the system, you should log out. Otherwise, you leave the system open for anyone to use. (However, both X Windows and HP VUE provide system "lock" features. You can lock the keyboard without logging out.)

If you see `login:` after leaving BASIC/UX, your system administrator has set up the system to log out directly from BASIC/UX. No further action is required.

If you are in the HP VUE environment, first make sure you have quit BASIC/UX, then move the mouse to the logout button in the Workspace Manager and click the left mouse button.

If you are in the X Window System, first make sure you have quit BASIC/UX, then type (Shift)-(CTRL)-(Reset) to exit the X Window System.

Once you have left BASIC/UX and X Windows, you may see the HP-UX system prompt ("$"). To log out from HP-UX, type (in lower case):

```
exit (Return)
```

You are logged out, and you should see the following:

```
login:
```

If you have problems exiting, see your system administrator.

3 Using BASIC/UX in the X Window System

The ability to create windows apart from your BASIC/UX root window allows you to send different types of data to various locations in your windowing environment. For example, one part of your program may send numeric results to one window and graphics drawings to another. This chapter covers the windowing operations that help you do tasks similar to these.

The window keywords covered in this chapter will only work if you are running your BASIC/UX system in the X Window environment. You will also need a mouse and Medium or High Resolution Monitor (e.g., HP 98785A display).

Windowing Operations with BASIC/UX

This section covers the following BASIC/UX windowing operations:

- Creating windows.
- Listing windows.
- Removing windows.
- Moving windows.
- Outputting graphics to a window.
- Clearing the contents of windows.
- Raising and lowering a BASIC/UX window in the window stack.
- Copying data between windows.

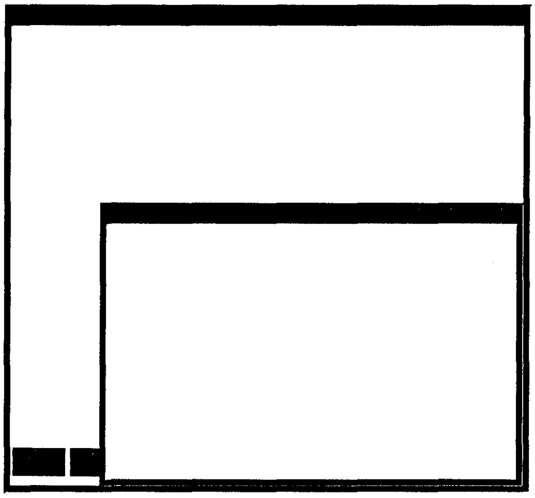
For a detailed explanation of the keywords used in this section, read the *HP BASIC Language Reference*.

Creating Windows

A window is a portion of the display that is accessible independently. The ability to access windows as independent printing or plotting devices makes them useful for programs that require the concurrent printing and plotting of numeric and graphics results. You can create one window for your numeric output and another for your graphics output.

The following example program called `create_w` (in `/usr/lib/rmb/demo`) creates a BASIC/UX window. Lines 110 through 120 assign the coordinates for the upper-left corner of the window being created. These are absolute coordinates from the display's upper-left corner. Lines 130 through 140 assign the window's width and height. The coordinates and window dimensions are used with the CREATE WINDOW statement to create window 601. The secondary keyword LABEL, when used with the CREATE WINDOW statement, allows you to assign useful names to each window you create.

```
100   INTEGER X_coor,Y_coor,Width,Height
110   X_coor=234  ! Assign the x coordinate position in pixels.
120   Y_coor=346  ! Assign the y coordinate position in pixels.
130   Width=640   ! Assign the width of the window in pixels.
140   Height=400  ! Assign the height of the window in pixels.
150   !
160   ! Create window number 601 and label it as window "One".
170   !
180   CREATE WINDOW 601,X_coor,Y_coor,Width,Height;LABEL "One"
190   END
```



Creating a Window

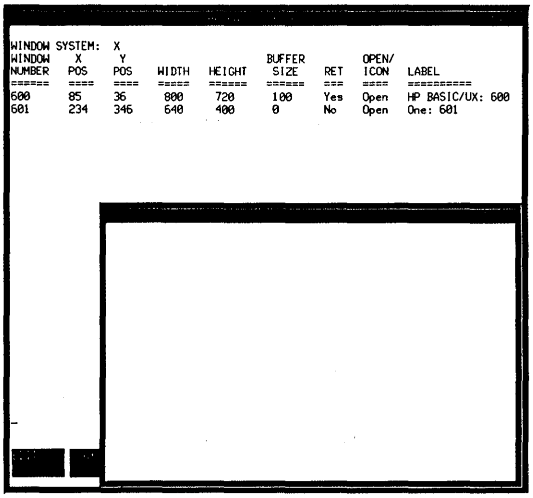
Listing Windows

A listing of the current windows is useful when you need to know a window's attributes. For example, if you needed to know which window numbers have been used or whether the window will retain a graphics image. To list the current windows, execute the following command:

```
LIST WINDOW (Return)
```

3

If you executed the program in the previous section called `create_w`, your display will look similar to this:



```
WINDOW SYSTEM:  X
WINDOW    X      Y                             BUFFER          OPEN/
NUMBER   POS    POS    WIDTH   HEIGHT   SIZE    RET   ICON   LABEL
======   ====   ====   =====   ======   ======  ===   ====   ==========
600      85     36      800     720      100    Yes   Open   HP BASIC/UX: 600
601      234    346     640     400      0      No    Open   One: 601
```

Listing of Current Windows and Their Attributes

The attributes are explained as follows:

`WINDOW NUMBER`	gives the window numbers of the currently active windows (e.g., 601).
`X POS`	gives the x coordinate position in pixels of the upper-left corner of the window listed.
`Y POS`	gives the y coordinate position in pixels of the upper-left corner of the window listed.
`WIDTH`	is the window's width in pixels.
`HEIGHT`	is the window's height in pixels.
`BUFFER SIZE`	shows the buffer size in lines. Note that lines on the screen can be scrolled into the buffer. These lines are in addition to the current lines in the root window.

RET	indicates with a Yes or a No whether the window will retain a graphics image. This is important if you output graphics information to a window that is obscured by another window and you want to bring it to the top of the "window stack" (for information on the window stack, read the subsequent section "Raising and Lowering a BASIC/UX Window in the Window Stack"). If the window was not retained, the image you would see in the window when you brought it to the top of the window stack would not be a complete one. To retain a window, use the secondary keyword RETAIN with the CREATE WINDOW statement. The secondary keyword RETAIN causes the raster image of graphics in a window to be saved in memory.
OPEN/ICON	shows whether the window you are looking for has been iconified (Icon) or is the full window size (Open).
LABEL	shows the name you have given to a particular window. This name was assigned when you used the secondary keyword LABEL with the CREATE WINDOW statement.

Removing Windows

When you are done with a window and its contents, the window can be removed to avoid clutter and confusion on the display. To do this, use either the keyword DESTROY WINDOW or SCRATCH W. The keyword DESTROY WINDOW can be used as a statement in a program or it can be executed from the keyboard line. Note that the keyword DESTROY WINDOW allows you to remove one window at a time. The following program called dest_w (found in /usr/lib/rmb/demo) uses the DESTROY WINDOW keyword to remove a window it creates.

```
100   CREATE WINDOW 601,200,250,640,400;LABEL "One"   ! Makes window 601.
110   !
120   WAIT 5
130   DESTROY WINDOW 601                              ! Remove window 601.
140   END
```

You can also remove the window by executing the SCRATCH W command. This command differs from the DESTROY WINDOW command as follows:

- It is not programmable.
- It destroys all created windows.

If you no longer need the BASIC/UX windows that you created using the keyword CREATE WINDOW and you want to remove them, execute the following command:

```
SCRATCH W (Return)
```

This command allows you to remove all of your BASIC/UX windows excluding the root window. The SCRATCH W command is convenient when you need to remove more than one window.

Moving Windows

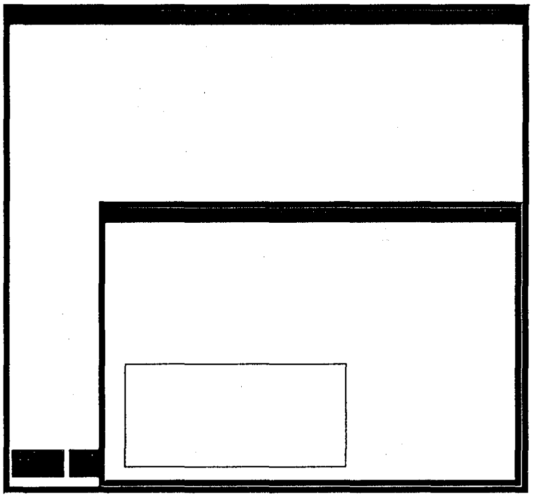
The MOVE WINDOW keyword allows you to move a window to another location on your display. The following program called `move_w` (found in `/usr/lib/rmb/demo`) shows how you can move a window horizontally across your screen using the keyword MOVE WINDOW.

```
100   INTEGER X_coor,Y_coor,Width,Height
110   X_coor=234   ! Assign the x coordinate position in pixels.
120   Y_coor=346   ! Assign the y coordinate position in pixels.
130   Width=640    ! Assign the width of the window in pixels.
140   Height=400   ! Assign the height of the window in pixels.
150   !
160   ! Create window number 601 and label it as window "One".
170   !
180   CREATE WINDOW 601,X_coor,Y_coor,Width,Height;LABEL "One"
190   !
200   ! Move the window horizontally across the screen.
210   !
220   FOR I=1 TO 100 STEP 2
230     MOVE WINDOW 601,X_coor+I,Y_coor
240   NEXT I
250   END
```

Outputting Graphics to a Window

A window can be assigned as a plotter for your graphics output. The PLOTTER IS keyword is used to assign a window as a plotting device. The following program called `plot_w` (found in `/usr/lib/rmb/demo`) uses the PLOTTER IS keyword to assign window 601 as the plotting device and draws a rectangle in that window.

```
100   INTEGER X_coor,Y_coor,Width,Height
110   X_coor=234  ! Assign the x coordinate position in pixels.
120   Y_coor=346  ! Assign the y coordinate position in pixels.
130   Width=640   ! Assign the width of the window in pixels.
140   Height=400  ! Assign the height of the window in pixels.
150   !
160   ! Create window number 601 and label it as window "One".
170   !
180   CREATE WINDOW 601,X_coor,Y_coor,Width,Height;LABEL "One"
190   !
200   ! Draw a rectangle in window 601.
210   !
220   GINIT
230   PLOTTER IS 601,"WINDOW" ! Assign window 601 to be the plotter.
240   MOVE 5,5                ! Move the pen to the starting
250                           ! position of the plot.
260   RECTANGLE 60,40         ! Draw the rectangle in window 601.
270   END
```



Sending Graphics Output to a Window

Clearing the Contents of Windows

You can clear the contents of a window using the CLEAR WINDOW keyword. This keyword can be used to clear the rectangle from the window created in the previous example. To clear window 601, type:

```
CLEAR WINDOW 601 (Return)
```

Note that the CLEAR WINDOW keyword is similar to the CLEAR SCREEN keyword; however, if you want to execute the CLEAR SCREEN command to clear a window, that window has to be the current PRINTER IS device. If you want to clear only graphics from a window, use the GCLEAR command while the window is the current PLOTTER IS device.

Raising and Lowering a BASIC/UX Window in the Window Stack

This section explains how to uncover a selected window and bring it to the top of the "window stack" and how to lower a window to the bottom of the "window stack." A "window stack" is several "windows" that are layered on top of each other.

CRT Control register 22 gives you a means for moving a window to the top or bottom of the window stack. The commands used to do this are as follows:

`CONTROL` *wind_num*`,22;1`	*raises a window to the top of the window stack*
`CONTROL` *wind_num*`,22;0`	*lowers a window to the bottom of the window stack*

The variable *wind_num*, in the above command, is a window number between 601 and 699. The value 1 when sent with the above command causes a designated window to be raised to the top of the window stack. If a value of 0 is used with the above command, the designated window is lowered to the bottom of the window stack.

The following program called `raise_w` (found in `/usr/lib/rmb/demo`) uses the keyword CONTROL to access register number 22 to move a window to the top of the window stack.

```
100   CREATE WINDOW 601,200,250,640,400;LABEL "One"   ! Makes window 601.
110   CREATE WINDOW 602,220,270,640,400;LABEL "Two"   ! Makes window 602.
120   CREATE WINDOW 603,240,290,640,400;LABEL "Three" ! Makes window 603.
130   !
140   WAIT 2          ! This allows you time to look at the current
150                   ! window stack.
160   CONTROL 601,22;1 ! Raise window 601 to the top of the window stack.
170   END
```

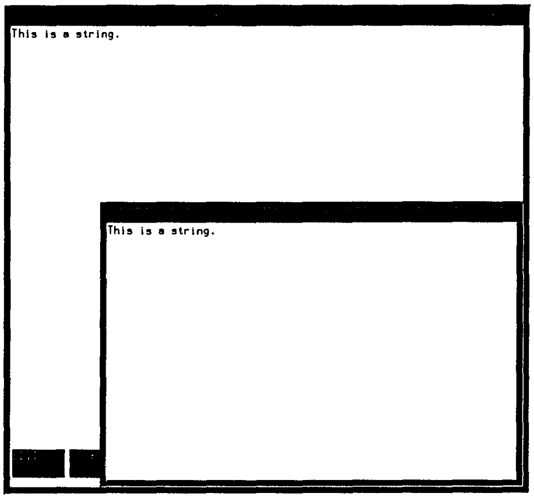
If you wanted to move window 603 to the bottom of the window stack, you would change line 160 in the above program to read as follows:

```
160   CONTROL 603,22;0 ! Push window 603 to the bottom of the window stack.
```

Copying Data Between Windows

The following program `tran_w` (found in `/usr/lib/rmb/demo`) copies the alpha contents of window 601 into the root window. To do this, window 601 is created and assigned as the printing device. Next, the OUTPUT statement in line 240 is used to send a string of characters to window 601. Lines 270 and 280 position the cursor in window 601 to the beginning of the string and line 290 reads that string into a string variable. The root window is then assigned as the printing device and the contents of the string variable are printed in window 600.

```
100   INTEGER X_coor,Y_coor,Width,Height
110   X_coor=200  ! Assign the x coordinate position in pixels.
120   Y_coor=350  ! Assign the y coordinate position in pixels.
130   Width=640   ! Assign the width of the window in pixels.
140   Height=400  ! Assign the height of the window in pixels.
150   !
160   ! Create window number 601 and label it as window "One".
170   !
180   CREATE WINDOW 601,X_coor,Y_coor,Width,Height;LABEL "One"
190   !
200   ! Send a string to window 601 and read that string back
210   ! into window 600 (the default window).
220   !
230   PRINTER IS 601          ! Assign window 601 to be the printer.
240   OUTPUT 601 USING "K";"This is a string."     ! Send string to
250                                                ! window 601.
260                                                !
270   CONTROL 601,0;1 ! Position cursor in column one of window 601.
280   CONTROL 601,1;1 ! Position cursor in row one of window 601.
290   ENTER 601 USING "K";String$                  ! Read string from
300                                                ! window 601.
310                                                !
320   PRINTER IS 600           ! Assign window 600 to be the printer.
330   PRINT String$            ! Print the string in window 600.
340   END
```


This is a string.

This is a string.

Copying Data Between Windows

Customizing the X Window System

You can change the default colors, location, border width, and buffer size on windows for BASIC/UX:

- If you are using HP VUE, refer to the *HP Visual User Environment User's Guide* for information about customizing the VUE environment.
- If you are using X Windows without HP VUE, refer to *Using the X Window System* for information about customizing the windows environment.

4

Introduction to the System

In addition to the complete set of manuals provided with HP BASIC, HP E2160A BASIC Plus is available from Hewlett-Packard for convenient on-line HELP.

This chapter explains simple BASIC/UX operations such as interpreting the display, typing commands, using printers, and using and redefining softkeys.

Significance of Letter-Case

Letter-case is important in BASIC. Keywords consist of all capital letters (for example, "BEEP"). Identifiers such as variable names, line labels, or subprogram names consist of an initial capital letter followed by lower-case letters or numbers. However, if you type *all capital* or *all lower-case* letters, the BASIC editor is *usually* "smart" enough to recognize what you mean from the context. For example, if you type "beep", and press Return, BASIC will execute the BEEP command. Don't type "Beep," or BASIC will decide that you meant to type a variable name called "Beep".

Let's look at an example using the program editor.

If you type:	**The editor enters:**	**Because:**
let abc=1	LET Abc=1	LET is a keyword and Abc is a variable name.
print "hello"	PRINT "hello"	PRINT is a keyword
BEGIN: beep	Begin: BEEP	Begin is a line label and BEEP is a keyword.

The resulting program lines will look like this:

```
10     LET Abc=1
20     PRINT "hello"
30 Begin:    BEEP
```

Note that literal strings ("hello") must be typed *exactly* as desired.

Letter-case is also important in HP-UX commands. EXECUTE "ls" is correct, while EXECUTE "LS" is not correct. See the *HP BASIC 6.2 Language Reference* for more information.

Many of the examples in this text do not begin with line numbers. Keep in mind, however, that when these statements are used within a BASIC/UX program they will be prefaced by a line number (see *HP BASIC Programming Guide*). For example:

```
10  MOVE WINDOW 603,100,400
```

Where "10" is the line number

Program Control

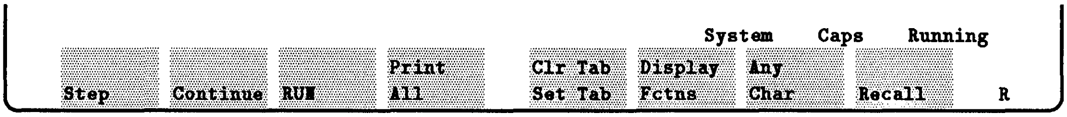
When BASIC is booted, it clears memory and assigns various default values. This condition is the **power-on state**. For a complete list of power-on defaults, see the "Useful Tables" section in the *HP BASIC 6.2 Language Reference*.

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

- Find out what your computer is doing.
- Gain control of your computer to use it properly.

The Status Indicators

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



System Status Indicators

Indicator	Description	Keyboard Control
Softkey Menu Indicator	The following labels are used depending on which menu is selected: `System`, `User 1`, `User 2`, `User 3`.	Select menu with System, User, or Shift-Menu.
Caps Lock Indicator	Indicates the keyboard's caps mode status.	Toggle caps mode using Caps.
Program Status Indicator	Indicates the run light status.	See following tables for run light indicator meanings and how to control system/program status.
Softkey Labels	Label the keyboard function keys' operations.	Turn on and off with KEY LABELS ON and OFF or Menu.
Run Light	Graphic indicator of the system or program status.	See following tables for run light indicator meanings and how to control system/program status.

The character in the lower right corner is called the **run light**. The following table shows the various run light indications and their meanings.

Run Light Indications

Status Indicator[1]	Run Light	System State
`Idle`	(*blank*)	Program stopped; can execute commands; CONTINUE not allowed.
`Running`	R	Program running; can execute commands; CONTINUE not allowed.
`Paused`	_	Program paused; can execute commands; CONTINUE *is* allowed.
`Transfer`	I	Program paused, but an overlapped TRANSFER (I/O) operation is still in progress; can also execute commands.
`Input?`	?	BASIC/UX 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.
`Execute`	E	A command is being executed in the HP-UX environment.
`Boot`	B	BASIC/UX is in the process of booting; all keyboard input is lost. The runlight displayed by consoles and X windows at boot-up is `Boot`, and the runlight displayed by terminals is `B`.

[1]Note that these indicators are displayed only if softkey labels are currently on. Use (Menu) or the KEY LABELS ON statement, to turn these labels on.

Is There a BASIC Program in Memory?

To see if a BASIC program is in memory, use the LIST command to print the program lines. For example:

```
PRINTER IS CRT (Return)
LIST (Return)
```

Tell BASIC to print to the CRT display.

Typical results:

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


Available memory = 5629926
```

If you don't want to wait for the entire program to list, you can stop it by pressing (Break). If there is no program in memory, LIST prints the amount of available memory.

Controlling Program Status

To pause or stop a program before its normal completion, continue operation, or abort an I/O statement, use the following keys:

Pausing and Stopping Programs

ITF Keyboard	Effect
Stop (Pause)	Pauses a program after it finishes the current line and any I/O in progress. Useful for pausing a program that is executing an INPUT statement, leaves internal information intact. You can resume program execution with `Continue` (or the CONT command).
f2 `Continue`	`Continue` (or the CONT command) after Pause causes program to resume in a normal manner from where it was paused.
Break (Clr I/O)	Cancels any I/O in progress (ENTER or TRANSFER) and pauses the program. The program counter returns to the beginning of the canceled I/O statement, so `Continue` resumes execution beginning with that same statement.
Shift-Stop (Stop)	Stops the program at the end of the current line, returning the program to the main context. Does not affect interfaces, CRT, program memory, variables, tabs, or the `Recall` (f8) buffer. `Continue` is not allowed after Stop.
Reset SHIFT-Break	The most drastic and complete way to stop a program. The program stops immediately, cancels I/O operations, closes open files, and resets all interface cards. However, the printout area of the CRT, program or variable memory, tabs, and the `Recall` buffer are not affected. `Continue` is not allowed after Reset.

ITF keyboard definitions are easy to remember if you use the BASIC/UX keyboard overlay and keep softkey labels turned on.

Determining Current System Devices

You can determine the current state of several system defaults by using the statements in the following table. See the *HP BASIC 6.2 Language Reference* for more a complete list of the system defaults or for information about SYSTEM$.

Note BASIC/WS provides several language extension binaries that can be loaded at the option of the user. However, *all* of these binaries are part of the BASIC/UX core system. You don't need to worry about whether a particular binary is present.

System Defaults

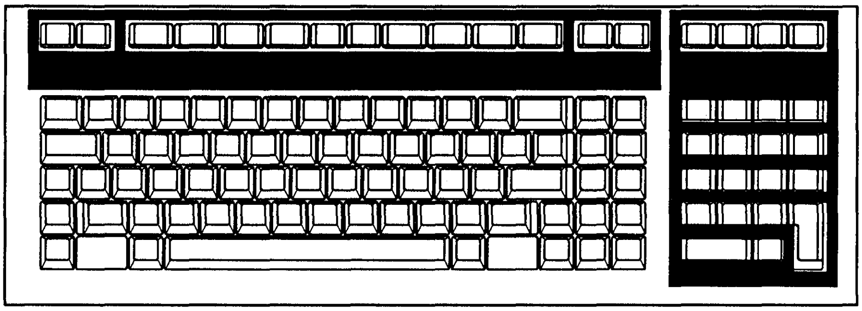
Method	Explanation	Default
`SYSTEM$("PRINTER IS")`	Returns the current system printer's select code (destination for PRINT operations).	`PRINTER IS CRT`
`SYSTEM$("PRINTALL IS")`	Returns the current printall printer's select code (destination for system messages when PRINTALL ON is active).	`PRINTALL IS CRT`
`SYSTEM$("DUMP DEVICE IS")`	Returns the current dump device's select code (destination of DUMP ALPHA and DUMP GRAPHICS).	`DUMP DEVICE IS 701`
`SYSTEM$("MSI")`	Returns the default mass storage device used when one is not explicitly specified.	device from which BASIC/UX booted
`SYSTEM$("AVAILABLE MEMORY")`	Returns the sum of the memory available for program storage, stack and COM.	*(not applicable)*

System Defaults (continued)

Method	Explanation	Default
SYSTEM$("VERSION: BASIC/UX")	Returns the BASIC/UX revision number.	(*not applicable*)
SYSTEM$("VERSION:*bin*")	Returns the revision number of *bin* ; for example: SYSTEM$("VERSION:EDIT")	(*not applicable*)
SYSTEM$("WILDCARDS")	Returns the current status of WILDCARDS: ■ OFF if disabled. ■ "UX:" if UX wildcards enabled without escape character. ■ "UX:\" if UX wildcards enabled with \ as the escape character. ■ "UX:'" if UX wildcards enabled with ' as the escape character. ■ "DOS:" if DOS wildcards enabled.	OFF
SYSTEM$("VERSION:OS")	Determines HP-UX version; A means single-user, B means multi-user.	(*not applicable*)

Using the Keyboard

The following section briefly describes keyboard use. For detailed information on using your ITF keyboard, see chapter 11, "Keyboard Information."



ITF Keyboard (with BASIC/UX Keyboard Overlay)

Use the keyboard to perform the following BASIC tasks:

- Perform calculations.
- Type and execute commands.
- Load and run programs, and control program execution.
- Type, edit, and store programs.

Performing Calculations at the Keyboard

You can use BASIC as a calculator to evaluate numeric expressions using the following arithmetic operators.

Arithmetic Operators for Keyboard Calculations

Operator	Operation	Example	Results
-	subtraction	`2-4` Return	`-2`
+	addition	`5.23+2.8-2` Return	`6.03`
/	division	`5+3/2-1` Return	`5.5`
*	multiplication	`3*3-1` Return	`8`
^	exponentiation	`3^2*2-2` Return	`16`
`SIN`, `COS`, etc.	functions	`SQRT(25)/5` Return	`1`
(...)	grouping	`SQRT(125/5)+(2*3)/4` Return	`6.5`

For example:

```
99/9        Characters you type appear here. Press Return
11          System response appears here.
```

For a complete explanation of all math operations, see the *HP BASIC Programming Guide* chapters on "Numeric Computation" and "Evaluating Scalar Expressions."

Typing and Executing Commands

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

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

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

Try the commands in the following table.

- Note the status indicator as you execute commands.
- When you use the `EXECUTE` command in the X Window System, the results are displayed in the HP-UX window from which BASIC/UX was invoked.

Example BASIC Commands

Type This Command	What It Does	Example Results
`DATE$(TIMEDATE)` (Return)	show BASIC date	`26 Jun 1988`
`TIME$(TIMEDATE)` (Return)	show BASIC time	`09:53:53`

`EXECUTE "who"` (Return)	display user names	`julian  ttyp0  Feb 29 08:57`
		`fred  tty02  Feb 29 08:32`

`KEY LABELS OFF` (Return)	turn off softkey labels	(look at the bottom of your screen or window; `KEY LABELS ON` or (Menu) restores the labels)
`SYSTEM$("VERSION:OS")`	determine HP-UX version; `A` means single-user, `B` means multi-user	`7.0B HP-UX`

You can set the time and date using SET TIMEDATE.

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

Note SET TIMEDATE sets only the BASIC/UX clock — it has no effect on the HP-UX clock.

When you make errors entering commands, you will receive an error message. Here are some common ones:

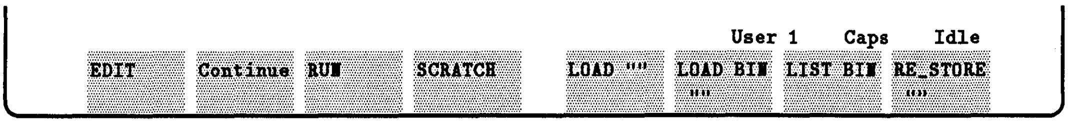
Possible Error Messages When Entering Commands

Error Message	Typical Cause of Error
`Error 910 Ident not found in context`	Mixed letter-case, or mistyped a parameter.
`Error 949 Syntax error`	Mistyped command (check spelling).
`rmb-execute: WHO: not found`	If you use the `EXECUTE` command, be sure to use the proper letter-case within the quote marks. In this example, `WHO` should be `who`.
nothing on display	If you used the `EXECUTE` command in X, all output goes to the HP-UX window from which the BASIC/UX window was started. Shuffle the HP-UX window to the top (see the previous chapter, "Using BASIC/UX in the X Window System").

Using Softkeys

Softkey Labels

If you are using the ITF keyboard, the following softkey labels are displayed at the bottom of the screen when you first turn on your computer or boot BASIC:



```
                                                User 1    Caps    Idle
 EDIT      Continue  RUN      SCRATCH      LOAD ""   LOAD BIN  LIST BIN  RE_STORE
                                                      ""                  ""
```

When you press a softkey, it produces a commonly used command, which is indicated by the key label. Some softkeys act simply as typing aids. For example, if you press EDIT (f1), the command EDIT appears on the command line. BASIC will go into EDIT mode when you then press Return. Other softkeys immediately execute the command. For example, if you press RUN (f3), the program currently in memory will be run—you don't have to press Return.

BASIC automatically defines what each softkey does, and what its key label is. However, you can redefine any softkey to execute commands specific to your needs. For further information refer to "Redefining Softkeys" later in this chapter.

Note There is an exception to the normal operation of a softkey. The softkey will execute the command indicated by its key label, except when that softkey has been defined by a running program to produce an interrupt. Refer to "Program Structure and Flow" in the *HP BASIC Programming Guide* for information on ON KEY interrupts.

On the ITF keyboard, if the softkey labels are not displayed, press Menu to display them. Press Menu again to turn them off. You can also turn the softkey labels on with either of the following commands:

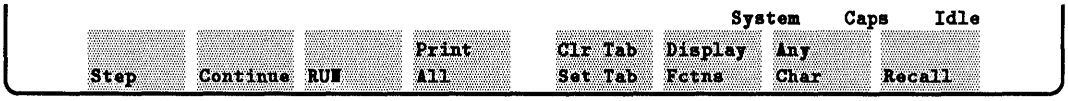
```
KEY LABELS ON Return

CONTROL CRT,12;2 Return
```

Selecting a Menu

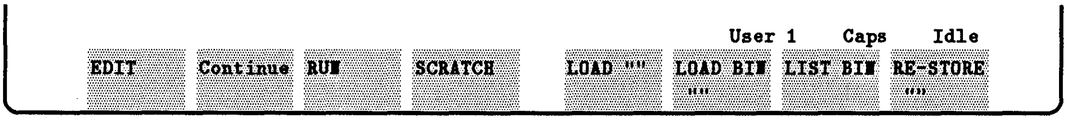
The set of eight softkey labels at the bottom of the screen is a **softkey menu.** The softkeys (f1 through f8) have four independent sets of definitions for the ITF keyboard. Select the menu that you want as follows:

Press System to display the System softkey menu:



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

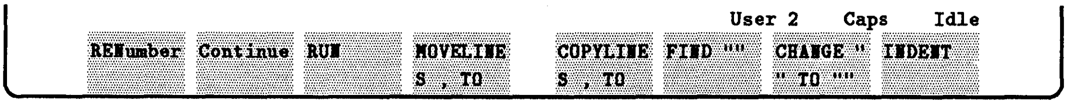
Press User to return to the User 1 softkey menu:



```
                                                          User 1    Caps     Idle
EDIT      Continue  RUN        SCRATCH     LOAD ""   LOAD BIN  LIST BIN  RE-STORE
                                                     ""                  ""
```

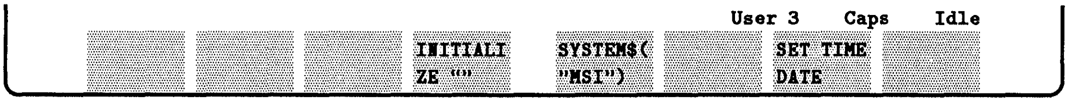
(The User 1 menu is the default menu at system power up.)

Press Shift-Menu to display the User 2 softkey menu:



```
                                                          User 2    Caps     Idle
RENumber  Continue  RUN        MOVELINE   COPYLINE  FIND ""   CHANGE "  INDENT
                               S , TO     S , TO              " TO ""
```

Press Shift-Menu again to display the User 3 softkey menu:



```
                                                          User 3    Caps     Idle
                               INITIALI   SYSTEM$(            SET TIME
                               ZE ""      "MSI")              DATE
```

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

Redefining Softkeys

This section describes how to create your own set of softkey definitions; it also shows how to store these definitions in a file so you can reload them at a later time.

An an alternative to the following procedures, you can write a program that defines the softkeys, using the SET KEY statement. See the "Communicating with the Operator" of the *HP BASIC 6.2 Advanced Programming Techniques* manual for details.

Memory Available for Softkey Definitions

BASIC/UX uses about 1024 bytes of memory to store the softkey definitions. Each definition can have:

- up to 256 characters on systems with high resolution displays
- up to 160 characters on systems with medium resolution displays.

Exceeding these limits results in lost characters.

Examples of Redefining Softkeys

Use the EDIT KEY command to redefine softkeys with your own softkey definitions. Don't worry about losing the original softkey definitions. You can get them back by executing LOAD KEY, or by rebooting. For programming purposes, softkeys are numbered 1 through 24. USER 1 keys are numbered 1-8, USER 2 keys are numbered 9-16, and USER 3 keys are numbered 17-24.

Example 1

This example defines a softkey that produces `My very own keystrokes`.

1. Enter the edit-softkey mode for the desired softkey:
   a. Press (User) until the softkey-menu indicator displays `User 1`.
   b. Press `EDIT` ((f1)), then press (f1) again. (f1) is the key you are going to define. Press (Return).

   If you are using an unmodified version of BASIC/UX, your display should look similar to this:

```
k#EDIT               Displayed on the keyboard input line
Editing key 1        Displayed on the system message line
```

2. Press (Shift)-(Clear line) to clear the key's current definition.
3. Type the desired characters on the keyboard input line.

   `My very own keystrokes`

4. Enter or cancel the softkey redefinition:
   a. To enter the softkey's definition and exit softkey editing mode, press (Return).
   b. To cancel the redefinition and retain the existing definition, press Stop ((Shift)-(Stop)).
5. If you entered the new definition, verify that the key works as desired.

   Press (f1). `My very own keystrokes` should appear.
6. Press (Shift)-(Clear line) to clear the line for the next example.

Example 2

This example redefines a softkey to do the following:

- Clear the line you are on
- Type a command
- Execute the command.

1. Here is another way to enter the edit-softkey mode. Since you redefined (f1) in the previous example, you'll have to enter the edit softkey mode by typing:

   ```
   EDIT KEY 2 (Return)
   ```

2. Press (Shift)-(Clear line)
3. Enter the following keystrokes:

   (CTRL)-(Shift)-(Clear line) `LIST` (CTRL)-(Return)

   The notation (CTRL)-(Return) means to hold down the (CTRL) key then press (Return). The (CTRL) key tells BASIC not to execute that key's function, but to enter that key in the softkey definition. The display will show an inverse-video k (shown here as k), followed by another character. For example, (CTRL)-(Return) produces kE.

4. To enter the softkey's definition and exit softkey editing mode, press (Return) To cancel the redefinition and retain the existing definition, press (Shift)-(Stop) (Stop).
5. If you entered the definition above, you can execute the LIST command by pressing (f2). You don't need to press (Return) because you already included it in the softkey definition.

Improving Softkey Labels

You may want to improve specific labels to fit in the label area on the display. The following example shows how to improve the (f2) label for the LIST command used in the previous example.

Example 3

1. Type:

   ```
   EDIT KEY 2 (Return).
   ```

2. To clear the current softkey definition, press (Shift)-(Clear line).
3. Type the following line (enter 12 spaces after LIST.)

   ```
   LIST            (CTRL)-(Shift)-(Clear line) (CTRL)-(Return)
   ```

4. Press (Return) to save.
5. Press (f2) to see how the new definition works. Notice that LIST is momentarily displayed, then cleared and executed.

Listing the Current Softkey Definitions

You can list all current softkey definitions by executing one of the following statements:

`LIST KEY`	*lists on the default printer (usually CRT)*
`LIST KEY #PRT`	*lists on printer, if available*
`LIST KEY #701`	*lists on device at 701, if available*

Since most printers cannot print the inverse-video k in softkey definitions, LIST KEY substitutes the letters `System key:` for this character. For example:

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

Storing and Loading Softkey Definitions

STORE KEY stores all of the current softkey definitions in a file. Use LOAD KEY to restore the default definitions or to load your own definitions back into the computer. The following examples show how to store and load softkey files and apply only to the currently specified mass storage volume.

`STORE KEY "MyKeys"`	*To store definitions in new file called* `MyKeys`
`RE-STORE KEY "MyKeys"`	*To replace definitions in an existing file of* `MyKeys`
`LOAD KEY "MyKeys"`	*To load definitions stored in file called* `MyKeys`
`LOAD KEY`	*To restore default softkey definitions*

The SCRATCH Commands

You can use the SCRATCH command to clear the BASIC/UX system's memory and restore default parameters. Let's look at what each form of this command will do. (For further information, refer to the *HP BASIC 6.2 Language Reference*.

`SCRATCH`	Clears all program lines currently in the BASIC/UX system's memory. It also clears all variables which are not in COM. See the "Subprograms" chapter of *HP BASIC Programming Guide* for a description of COM.

`SCRATCH A`
`SCRATCH ALL`
	Clears most everything from the BASIC/UX system's memory, restoring the system to its default state. The only exceptions are the `Recall` (f8) key's buffer and the real-time clock.

`SCRATCH C`
`SCRATCH COM`
	Clears *all* variables from the BASIC/UX system's memory, including COM. However, the current program and softkey definitions are left intact.

`SCRATCH KEY`	Clears softkey definition(s). See the descriptions of softkeys in preceding sections of this chapter for further information.

`SCRATCH R`
`SCRATCH RECALL`
	Clears the `Recall` (f8) key's buffer.

`SCRATCH W`
`SCRATCH WINDOW`
	This command only can be executed from a window system. Otherwise, an error occurs. This command destroys all windows *created from BASIC/UX*. It does not destroy the BASIC/UX window itself.

5

BASIC/UX Mass Storage Concepts

This chapter covers some general mass storage concepts with emphasis on how mass storage is used in BASIC/UX. 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. Common mass storage devices include the following:

- Hard disk drives.
- Flexible disk drives.
- Tape drives.

The most common devices are the hard and flexible disk drives. Flexible disks can store on the order of several thousand bytes of data. On the other hand, hard disks can store up to hundreds of millions of bytes. To keep this large amount of data well organized and accessible, mass storage is organized into files, volumes, and directories.

There are basically two types of mass storage organization structures:

- In a *hierarchical* directory structure, program and data files are organized in a hierarchy of directories and subdirectories, starting at the root directory. Hierarchical directories are discussed in detail in the following section.
- In a *non-hierarchical* structure, a flexible disk or hard disk drive unit is organized into one or more separate volumes. An example of such a structure is the Logical Interchange Format (LIF) system. Each LIF volume has a single LIF directory that lists all of the files in that volume. There are no subdirectories. LIF disk media are covered later in this chapter.

Hierarchical Directories

A directory contains information about files, such as file name, size, and type. 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/UX directory formats that implement hierarchical directories:

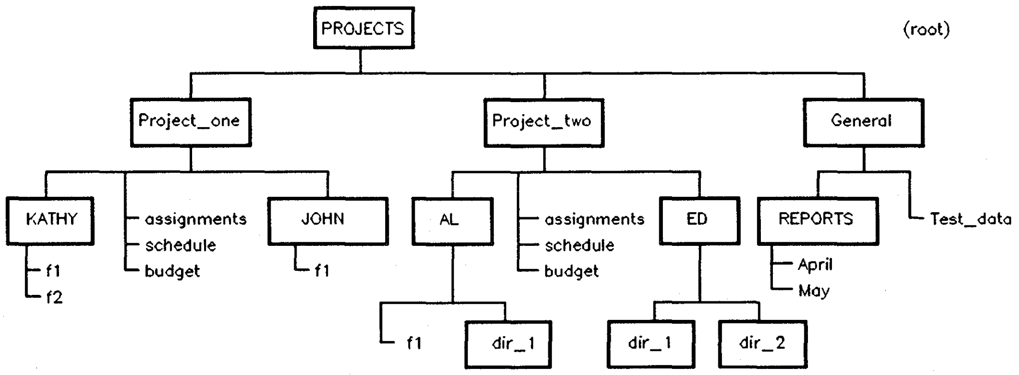
- Hierarchical File System (HFS) format (used with HP-UX, some BASIC, and some Pascal systems). (This format is also used by SRM/UX.)
- Shared Resource Manager (SRM) format. The disk format is actually called Structured Directory Format (SDF) on catalog listings of these directories.

What Is a Hierarchy?

As the word **hierarchy** suggests, hierarchical directories are arranged in levels. Such a 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 **subordinate** to the directory containing it.

In the following figure, 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 directory**. You cannot create a directory at a higher level than the root level.



Hierarchy of Directories

Uses of the Hierarchy: An Example

Suppose you're managing several projects, and each needs to access a shared disk. To organize the files for each project separately, you can create a directory for each project (as shown in the previous figure). Within each project directory, you can have a directory for each person working on the project, and so on.

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. For example, the file `budget` in the `Project_one` directory is distinct from the file `budget` in the `Project_two` directory.

To maintain security, BASIC/UX provides the capability of protecting access to directories and files. For example, you may want to allow only members of a project team to read that project's files. Or, you may want to prevent other users from altering the contents of a personal file. See "Protecting Files" in chapter 6, "Using Directories and Files."

Referring to Directories and Files in the Hierarchy

To access a directory or file, 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 (/).

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 one of these locations:

- The root directory.
- The current working directory.

The **current working directory** is the directory specified by the most recent MASS STORAGE IS statement. The *HP BASIC 6.2 Language Reference* discusses the rules for specifying HFS, SRM, and SRM/UX files and directories.

Choosing a Directory Format

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

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

The following recommendations will help you choose a format:

- *Use HFS format with hard disks.* You can access the hierarchical file system on your hard disk through the HP-UX operating system. This will give you optimum performance in the BASIC/UX environment. Refer to "Accessing Hard Disks Through HP-UX" later in this chapter. (This format is also used by SRM/UX.)
- *Use LIF format with flexible disks.* The LIF format will allow you to share disks with HP Series 200/300 BASIC workstations. For further information, refer to "Accessing LIF Media" later in this chapter.
- *Use SDF on an SRM system* if you want to share a disk between several workstations.

Accessing Hard Disks Through HP-UX

You can access your BASIC/UX system's hard disk by using the HFS file system through HP-UX. However, there are a few tasks that your system administrator must perform before you can do this. These tasks are described in detail in the *HP-UX System Administration Tasks* manual. However, the tasks are listed here for convenience:

1. Connect the hard disk to the HP-UX system.
2. Initialize the hard disk in HP-UX format using `/etc/mediainit`.
3. Create the HFS file system using `/etc/newfs`. (To use `/etc/newfs` there must be an entry for the hard disk in the `/etc/disktab` database.)

4. Mount the file system using `/etc/mount`.

Once the file system is mounted, you can access the file system using standard HFS directory path names, either from HP-UX or from BASIC/UX. For further information, refer to chapter 6, "Using Directories and Files."

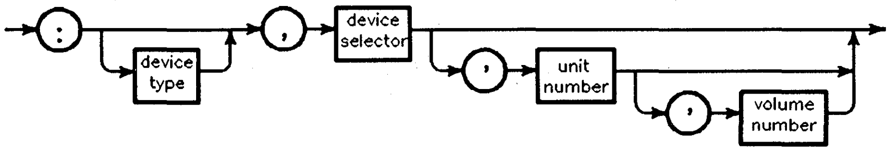
Accessing LIF Media

The traditional mass storage format for HP Series 200/300 Workstation BASIC (BASIC/WS) is the Logical Interchange Format (LIF). LIF media are formatted into one or more *volumes*, but there is no hierarchical directory structure. The LIF format is a very practical format for flexible disks since they hold relatively small amounts of data. However, for most hard disks it is much more practical to use one of the hierarchical directory structures covered previously in this chapter.

Specifying a LIF Volume

You can specify a LIF volume by means of an **msvs**, or mass storage volume specifier. (The msvs is sometimes called an msus, or mass storage unit specifier.)

The msvs has the following syntax:



Syntax of a Volume Specifier

Examples

```
:CS80,700
:,700

:HP9122,702,1
:,702,1

:SCSI,1400,1
:,1400,1
```

5

In each case the **device type** ("CS80", "HP9122", or "SCSI") can be left out. If it is, BASIC will determine the device type automatically. Thus, the "shortened" version works in each of the above examples.

The following table describes each part of the volume specifier.

Volume Specifier Components

Component	Explanation
Device type	Identifies the mass storage device's type. Once BASIC determines the device type, it can also determine device capacity, 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 determines type automatically. `CS80` — Any disk in the general class of "Command Set/80" devices (such as the HP 9122, and most other newer drives). `SCSI` — A built-in, 3 1/2-inch, flexible-disk drive (e.g. Models 362 and 382). For a list of all device types, see the *HP BASIC 6.2 Language Reference* entry for MASS STORAGE IS.
Device selector	Identifies the interface's select code (4, and 7 through 31) and primary address (HP-IB and SCSI devices only). Here are examples: `700` — Specifies select code 7 and primary address 0 (note that device selectors with HP-IB and SCSI addressing must contain 3 or 4 digits). `1402` — Specifies select code 14 and primary address 2.
Unit number	Tells BASIC 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 disk. For instance, the unit number of the right drive of an HP 9122 is 1, the left drive is 0 (internal drives of Model 236 computers are numbered in the opposite order).
Volume number	Identifies the volume number (multi-volume hard disks only, such as HP 9133X drives).

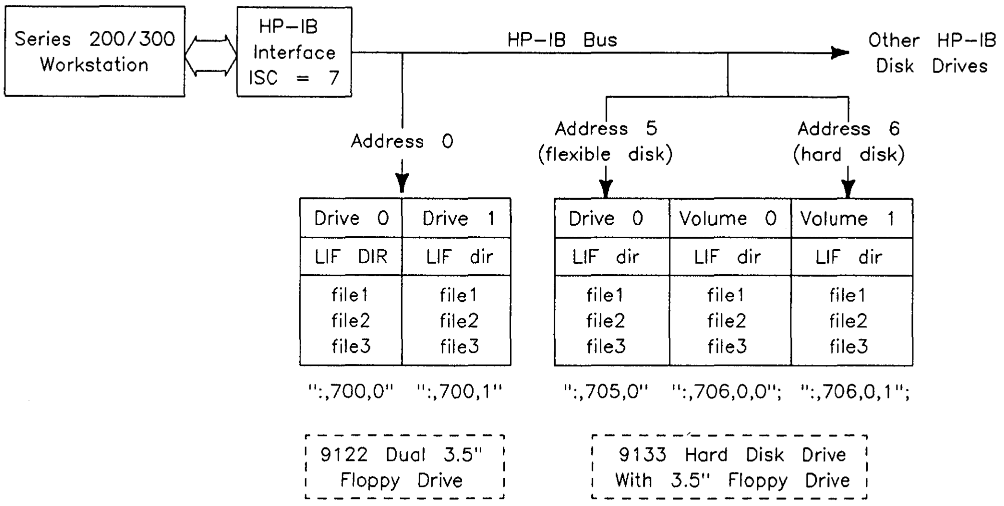
If you need to access LIF devices from your HP-UX system, have your system administrator install the devices and identify each device with a label showing its msvs.

For the BASIC/UX system, the most common use of LIF devices is to access LIF flexible disks from a LIF disk drive (for example, an HP 9122). LIF flexible disks provide a convenient means of transferring data files between your BASIC/UX system and HP Series 200/300 BASIC workstations. You can also access LIF hard disk volumes (for example, on an HP 9133), from BASIC/UX.

Note You can access a LIF formatted hard disk from BASIC/UX using its msvs. (For example, you could access an HP 9133 Hard Disk Drive moved to the BASIC/UX system from a BASIC workstation.) However, for optimum performance, use the Hierarchical File System (HFS) and access your hard disk through the HP-UX operating system. For further information, refer to "Accessing Hard Disks Through HP-UX."

A typical configuration is shown below, representing:

- an HP 9122 Disk Drive containing two 3 1/2 inch flexible disk drives.
- an HP 9133 Disk Drive consisting of one hard disk drive, partitioned into two volumes, and one flexible disk.



The volume specifiers for the HP 9122 Disk Drive are:

`:,700,0` and `:,700,1`	The drive type is "CS80", but is omitted. The HP-IB interface select code is "7". The primary address of the drive is "00". The left drive unit number is "0", and the right drive unit number is "1".

The volume specifier for the HP 9133 Disk Drive flexible disk drive unit is:

`:,705,0`	HP-IB interface select code is "7". Primary address of the drive is "05". The drive unit number is "0".

The volume specifiers for the HP 9133 Disk Drive hard disk are:

`:,706,0,0` and `:,706,0,1`	HP-IB interface select code is "7". Primary address of the drive is "06". The hard disk drive unit number is "0". The first volume number is "0". The second volume number is "1".

Initializing a LIF Flexible Disk

Before you can use a blank flexible disk, it must be initialized. You can initialize a LIF flexible disk from BASIC/UX by using the INITIALIZE statement in the same manner as for BASIC/WS. The INITIALIZE statement formats the disk and creates a LIF directory on it. You will need to have a LIF flexible disk drive connected to your BASIC/UX system, and you will need to know its msvs.

To initialize a LIF flexible disk, follow these steps:

1. If the disk is write-protected, write-enable it.
2. Determine the contents of the disk by executing the CAT statement. For example:

   ```
   CAT ":,700,0"
   ```

   This accomplishes two things. First, it ensures that you are using the correct msvs for the flexible disk drive. If you use the msvs of a hard disk volume by mistake, you will destroy the contents of that volume! Second, the CAT listing indicates the contents of the disk to be initialized. If you are initializing a previously used disk, you can check to make sure you won't lose any valuable files.
3. Execute the INITIALIZE statement. For example:

   ```
   INITIALIZE ":,700,0"
   ```

   Normally, you should INITIALIZE a disk with the default parameters. However, if you want to change the parameters, refer to the *HP BASIC Language Reference* manual for complete information about the INITIALIZE statement.

6

Using Directories and Files

Creating and Using Hierarchical Directories

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

HFS File Names

The disk on which BASIC/UX is installed, is HFS-formatted (Hierarchical File System).

HFS file names can be up to 255 characters in length for Long File Name systems (LFN) or up to 14 characters for Short File Name systems (SFN).

- Don't use control characters that might "confuse" your terminal.
- Avoid creating file names the same as system commands or file names.
- The CAT command (listing the files in a directory) truncates file names longer than 14 characters.

You can use either upper- or lower-case letters to name a file, however, it is case sensitive. For example, if you try to retrieve a file you named `SAM` by typing `sam`, the computer will not recognize it.

Determining Your Place in the Structure

To learn where you are in the directory structure, type one of the following commands and press (Return)

Determining Your Place in the Structure

Type this ...	What it does	Example Results
`EXECUTE "echo $HOME"`	Prints your HOME directory (displays in HP-UX window when running in X Windows)	`/users/arnie`
`SYSTEM$("MSI")`	Prints your current directory. All characters before `:HFS` (or `:REMOTE`) comprise the path name.	`/users/arnie/project1:HFS`

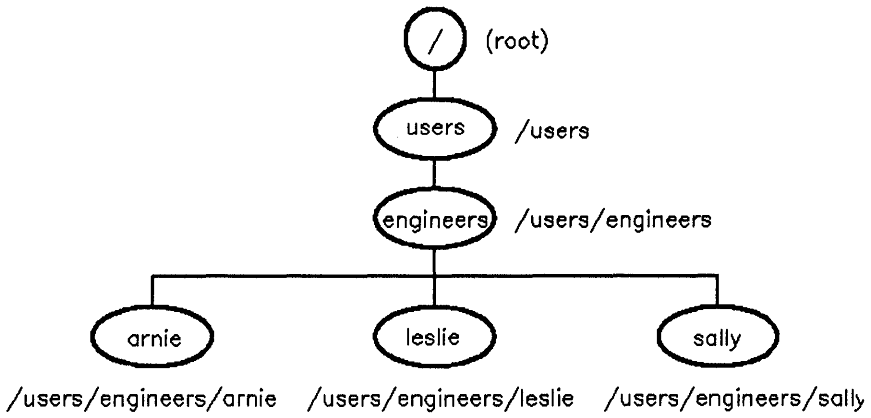
Referencing Files and Directories: Path Names

Since files are located all over the directory structure, you need a way to reference them. **Path names** show the computer a way to get to a particular directory or file.

- **Absolute path names** show the path to a directory or file starting from the root directory (the uppermost directory, symbolized by "/").
- **Relative path names** show the path to a directory or file from the current directory.

Using Absolute Path Names

The figure below shows the absolute path name for each directory in its structure. Notice that "/" separates directory names, and the first "/" indicates the root directory.

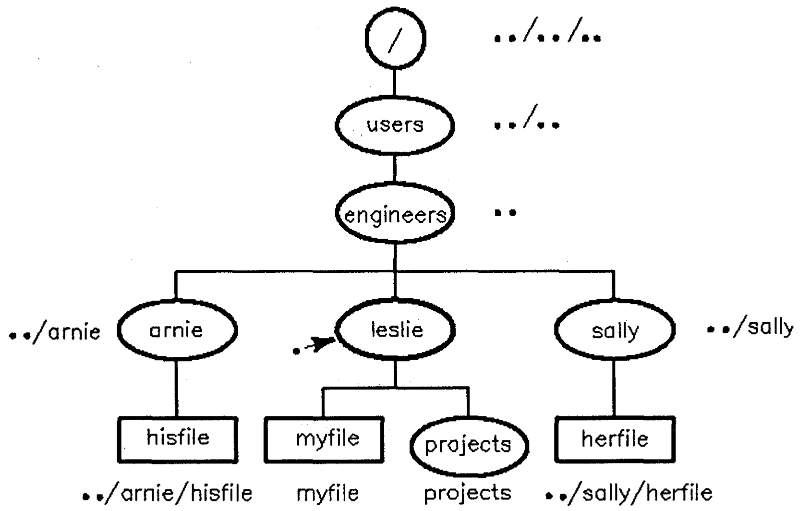


Absolute Path Names

Using Relative Path Names

The following figure shows relative path names from the directory **leslie** (`..` indicates the "parent" directory; `.` indicates the current directory).

Start at directory **leslie**. Moving up in the structure adds `..` to the path.



Relative Path Names from /users/engineers/leslie

For example, the path from `leslie` to `herfile` requires you to go up the structure to `engineers` (`..`), horizontally to `sally` (`../sally`), then down to `herfile` (`../sally/herfile`).

Try the example commands in the table below to list the contents of directories from BASIC/UX.

Example CAT Commands Using Absolute and Relative Paths

Type this ...	What it Does
`CAT` Return	List contents of current directory
`CAT ".."` Return	List contents of parent directory (relative)
`CAT "/users"` Return	List contents of `users` directory (absolute)
`EXECUTE "ls"` Return	Use the HP-UX `ls` command to list the current directory.

Understanding CAT Listings for Hierarchical Directories

Listing HFS Directories

When you list the contents of a directory, you see a list similar to the HFS listing below:

```
/users/leslie:HFS
LABEL:
FORMAT: HFS
AVAILABLE SPACE:    166780
                 FILE   NUM   REC     MODIFIED
FILE NAME        TYPE   RECS  LEN DATE         TIME PERMISSION OWNER GROUP
============== ===== ====== ===== ========= ===== ========== ===== =====
.profile         HP-UX   760     1 23-Jun-88 11:18  RWX------    204    10
.x11start        HP-UX  1729     1 23-Jun-88 11:05  RWX------    204    10
.Xdefaults       HP-UX   456     1  8-Jul-88  7:29  RWX------    204    10
reports          DIR       7    32 19-Jul-88 13:13  RWXRWXRWX    204    10
backup           DIR      39    32 27-Jun-88 13:41  RWXRWXRWX    204    10
notes            HP-UX   106     1 14-Jul-88 16:28  RW-RW-RW-    204    10
SPL_PROGRAM      PROG      1   256 14-Jul-88 16:28  RW-RW-RW-    204    10
ASCIIPROG        ASCII     1   256 14-Jul-88 16:28  RW-RW-RW-    204    10
```

Here is what each column means:

`FILE NAME`	Is the name of the file (up to 14 characters).
`FILE TYPE`	Indicates the file type (PROG, ASCII, BDAT, HP-UX, DIR).
`NUM RECS`	Is the number of records in the file.
`REC LEN`	Is the record length used in the file (file size is the product of RECORDS x LENGTH).
`MODIFIED DATE TIME`	Shows the date and time of day when the file was written or last modified.
`PERMISSION`	Shows who is allowed to R(ead), W(rite), or eX(ecute) the file for the file's Owner, Group, or Other (everyone on the system). Execute permission is also referred to as SEARCH permission.
`OWNER`	Numerical ID of the file's owner.
`GROUP`	Numerical ID of the file's group.

A CAT of an HFS directory requires R (read) and X (search) permissions on the directory, as well as X (search) permissions on all parent directories. See "Permitting HFS File Access" for information on permissions if you receive an error when executing `CAT` (for example, `ERROR 183 Permission denied`).

Listing SRM Directories

A typical SRM listing would look something like this example:

```
USERS/STEVE/PROJECTS/DIR1:REMOTE 21,0
LABEL:    Disk1
FORMAT:   SDF
AVAILABLE SPACE:    54096 SYS  FILE   NUMBER   RECORD    MODIFIED      PUB  OPEN
FILE NAME             LEV TYPE  TYPE  RECORDS   LENGTH DATE       TIME  ACC  STAT
=================== === ==== ===== ======== ======== ============== ==== ====
Common_data           1       ASCII       48       256  2-Dec-83 13:20 MRW  OPEN
Personal_data         1 98X6  BDAT        33       256  2-Dec-83 13:20      LOCK
Program_alpha         1 98X6  PROG        44       256  3-Dec-83 15:06  RW
HP9845_DATA           1 9845  DATA        22       256 10-Oct-83  8:45  R
HP9845_STORE          1 9845  PROG         9       256 10-Oct-83  8:47 MRW
Pascal_file.TEXT      1 PSCL  TEXT        37       256 11-Nov-83 12:25 MRW
Program_500           1 9000  PROG        12       256 13-Dec-83  9:54 MRW
```

Here is what each column means:

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)
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	Shows 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 indicates that Manager and Read capabilities are public, while other rights are protected requiring a password to access them. See the section "SRM Passwords and Locks" later in this chapter for details.

Listing SRM/UX Directories

A typical SRM/UX listing would look something like this:

```
:REMOTE 21,0
LABEL:  BOOT
FORMAT: SRM-UX
AVAILABLE SPACE: 123456789
                 FILE   NUMBER  REC     MODIFIED                                  OPEN
FILE NAME        TYPE  RECORDS  LEN  DATE          TIME    PERMS     OWNER GROUP STAT
=============== ===== ======== ===== =============== ========= ===== ===== ====
SYSTEMS          DIR        11    24  1-Mar-90 16:56 RWXR-XR-X     0     1
console          CDEV        0     1 12-Oct-90 17:05 RW--W--W-     0     1
EDITTEST.TEXT    TEXT        8   256 12-Dec-89 15:20 RW-R--R--   175    54
AUTOST           PROG        2   256  5-Jan-90 15:07 RW-R--R--   175    54
srmdpipe         PIPE        0     1 12-Oct-90 11:45 RW-------     0     1
PTEST            ASCII       1   256  2-Jan-90 10:51 RW-RW-RW-    17     9 LOCK
PTESTCAT         HP-UX     984     1  2-Mar-90 15:12 RW-RW-R--   175    54 OPEN
```

Here is what each column means:

FILE NAME	lists the names of the files and directories in the directory being cataloged.
FILE TYPE	indicates the file type. File types recognized by BASIC on SRM/UX are the following:

DIR - directory
PROG - BASIC program file
PIPE - named pipe

The SRM/UX user can also see the following special HP-UX files in a CAT listing, but cannot manipulate them:

NET - network special file
SOCK - HP-UX socket
BDEV - block special file
CDEV - character special file

If the system does not recognize a file type, it prints a numeric code or "OTHER".

NUMBER RECORDS	indicates the number of records in a file.
REC LEN	indicates the number of bytes in each file record (always 24 for directories (DIR), regardless of actual size).
MODIFIED DATE/TIME	(80-column format only) shows the date and time when the file's contents were last changed.
PERMS	specifies who has access rights to a file. R - indicates that a file can be read. W - indicates that a file can be written. X - indicates that a directory can be searched (meaningful for directories only). Three classes of user permissions exist for each file: OWNER - left-most three characters. GROUP - center three characters. OTHER - right-most three characters.

Listing Only File Names

The following statement produces a multi-column listing of the file names in the current working directory of the current default volume:

```
CAT; NAMES
```

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

Cataloging Selected Files

You can specify which files to list using the following options to CAT:

`CAT; SELECT "ABC"`	Lists only the files beginning with the specified letters "ABC".
`CAT "*A*"`	If you enable WILDCARDS, you could use this statement to list all file names containing the letter A. See the subsequent section on "Using Wildcards."
`CAT; COUNT Num_files`	Stores the number of lines in the catalog in a numeric variable called `Num_files`. (This variable must be defined in the current program or subprogram context before it can be used.)
`CAT; NO HEADER, SKIP 10`	Suppresses the catalog heading and skips the first 10 files in the directory.

See the *HP BASIC 6.2 Language Reference* for a complete description of these options.

Cataloging Individual PROG Files

A catalog of a PROG file yields the following additional information about the file:

- 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 requires only 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
```

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

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

See the *HP BASIC 6.2 Language Reference* for more detail on CAT. See *A Beginner's Guide to HP-UX* for more on path names.

Creating Directories

To organize your own files, you can create additional directories from your home directory with the `CREATE DIR` command.

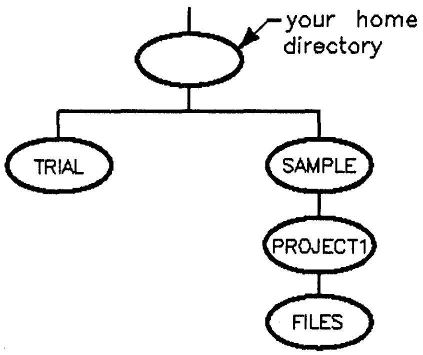
Generally, you should only create directories subordinate to (below) your home directory unless you are the system administrator.

Creating Directories from Your Home Directory

Try creating the examples from your home directory—we'll use the directories in a subsequent module for learning how to move from directory to directory.

- `CREATE DIR "SAMPLE"` Return
- `CREATE DIR "TRAIL"` Return
- `CREATE DIR "SAMPLE/PROJECT1"` Return
- `CREATE DIR "SAMPLE/PROJECT1/FILES"` Return

When you complete these commands, you create a directory structure shown in the following figure.



An Example Directory Structure You Create

Creating Directories with Absolute Path Names

If you are the system administrator, you may want to create directories outside your home directory. When you use absolute path names to create a directory, only the last directory name in the path can be the new directory. For example, if you type:

```
CREATE DIR "/tmp/sample"
```

`/tmp` *must* already exist (you'll see `ERROR 56 File name is undefined` if it doesn't exist).

You must have the correct permissions for each level of directories in the path down to the parent of the directory you are creating. See the subsequent section on "Permitting HFS File Access" for more information on permissions.

See the *HP BASIC 6.2 Language Reference* for more on the `CREATE DIR` command.

Changing Directories

To make another directory your current directory use the `MSI` statement (the "MASS STORAGE IS" statement).

Before trying the examples in this module, find the path name for your current directory so you can return to it. Type:

```
SYSTEM$("MSI")
```

For example, `/users/leslie:HFS`. The path name is: `/users/leslie`

Changing Directories with Relative Path Names

If you have to move to a directory in "close proximity" to your current directory, you can use a relative path name. The following table lists some commands you can try from your current directory based on the example from the previous section on "Creating Directories with Absolute Path Names".

After each command, type: `CAT` and check the path name at the top of the list. Then type: `CLS` to clear the screen and make it easier to see the next list.

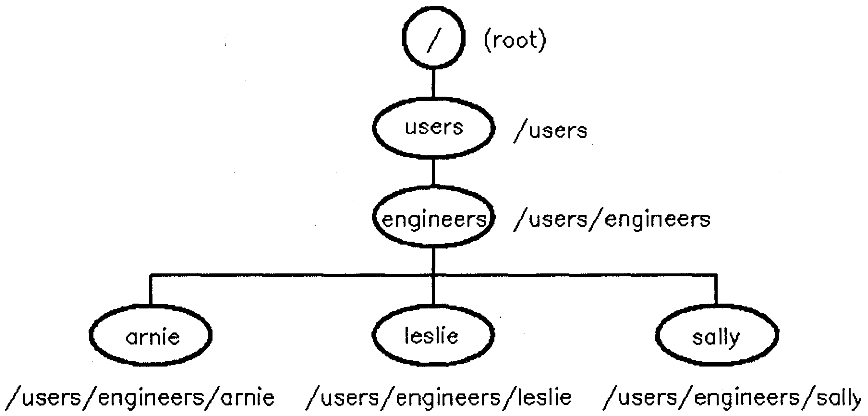
Changing Directories with Relative Path Names

Example	Explanation
`MSI "TRIAL"`	Move "down" to `TRIAL` directory.
`MSI "../SAMPLE"`	Move "up" to home directory and then "down" to `SAMPLE` directory.
`MSI "PROJECT1/FILES"`	Move down two levels to `FILES` directory
`MSI "../../.."`	Move up three levels to the directory where you started.

Type the `SYSTEM$("MSI")` command again to make sure you are in your home directory. If you aren't, type the `MSI` command with your home directory path name. For example: `MSI "/users/leslie"`

Changing Directories with Absolute Path Names

The table below shows some examples that use directories from the following figure (these examples won't necessarily work on your particular system).



Absolute Path Names

Changing Directories with Absolute Path Names

6

Example	Explanation
`MSI "/"`	Move directly to the root directory.
`MSI "/users/engineers"`	Move directly to **/users/engineers**.
`MSI "/users/engineers/arnie"`	Move directly to the **arnie** directory.

Changing Directories to LIF Disks

You can move to a directory for an LIF disk by including the mass storage volume specifier (*msvs*; see the previous chapter) of the LIF disk (see your system administrator for the disk's msvs). For example:

```
MSI ":,700"
```

where :,700 is the msvs. To return to the HFS directory from LIF disk, type:

```
MSI ":HFS" Return
```

If you see `Permission denied`, it means you tried to get into a directory that has been protected. The "owner" of the directory must change the permissions before you can change to that directory.

To determine your home directory's path name, type:

```
EXECUTE "echo $HOME"
```

See the *HP BASIC 6.2 Language Reference* for more information on the MSI keyword.

LIF Catalogs

Here is a typical catalog listing of a LIF directory. Displays with 80 columns or more, have two extra fields for 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, B9836).
`FILE NAME`	Lists the file names 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 (on 80-column or wider displays).
`TIME`	Time when file was last modified (on 80-column or wider displays).

Using Wildcards

BASIC lets you use wildcards with file system commands. **Wildcards** is the name given to a set of rules for using expressions as substitutes for file names. For instance, using an asterisk (*) as an argument in a file system command means to execute that command with all the matching files in the current directory. Wildcards can process multiple files by using file name expansion. Wildcards can also reduce typing by using file name completion. BASIC/WS wildcards are similar to wildcards used in the HP-UX operating system, and several options in addition to the asterisk are available. For complete information, see the entry for WILDCARDS in the *HP BASIC 6.2 Language Reference*.

Enabling and Disabling Wildcards

WILDCARDS is disabled at power-up, and after SCRATCH ALL. Wildcard processing must be explicitly enabled.

`WILDCARDS UX;ESCAPE "\"`	*Enable HP-UX style wildcards*
`WILDCARDS OFF`	*Disable wildcard recognition*

File Name Expansion

Certain file system commands can perform operations on multiple files when WILDCARDS are enabled. These commands are PURGE, COPY, LINK, CHGRP, CHOWN, PERMIT, PROTECT (on SRM files only), and CAT. All files that match a wildcard argument are processed by the command. For example, you could use wildcard arguments to process the files in the following SRM catalog listing.

```
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
================ === ==== ===== ======== ======== =============== === =====
AGENDA            1       ASCII     254        1 24-Apr-90 12:01 RW
MEMOS             1       DIR        12       24 22-Mar-89 23:12 RW
DATA1             1  98X6 BDAT        8      256  1-Apr-90 14:12 RW
DATA2             1  98X6 BDAT        7      256  2-Apr-90 14:12 RW
DATA3             1  98X6 BDAT        8      256  3-Apr-90 14:12 RW
DATA4             1  98X6 BDAT        5      256  4-Apr-90 14:12 RW
DATA5             1  98X6 BDAT        4      256  5-Apr-90 14:12 RW
DATA6             1  98X6 BDAT        8      256  6-Apr-90 14:12 RW
DATA7             1  98X6 BDAT        6      256  7-Apr-90 14:12 RW
DATA8             1  98X6 BDAT        5      256  8-Apr-90 14:12 RW
```

Note PURGE "DATA*" will erase all files in the directory path designated that begin with DATA (eg. DATA1, DATA2, DATA3). Before you execute a command using a wildcard, be certain you want to affect all files designated.

To copy all of the files prefixed by DATA to /PROJECTS/Project_one/JOHN, type:

```
COPY "DATA*" TO "/PROJECTS/Project_one/JOHN"
```

If an exception occurs while processing one of the files, the file name is displayed followed by a warning message. The command continues processing any remaining files that match the wildcard argument. After the command is finished processing the files, `ERROR 293 Operation failed on some files` is generated. Only one error is generated for a command no matter how many warning messages are generated for that command.

If multiple warning messages are generated, each message writes over the previous message. To avoid missing any warning messages, press PRINTALL. PRINTALL sends all warning messages and their corresponding error message to the PRINTALL IS device.

File Name Completion

Certain file system commands let you use file name completion with wildcards. Instead of specifying a complete file name as the argument for a command, you can specify a wildcard expression that matches a desired file name. The command finds the matching file name, then processes the file exactly as if you had specified the file name explicitly. File name completion reduces typing required for long file names.

Commands that allow file name completion are ASSIGN, DICTIONARY IS, GET, GFONT IS, LOAD, LOAD KEY, LOAD SUB, MSI, PROTECT(LIF files only), RENAME, RE-SAVE, RE-STORE, and RE-STORE KEY.

Using the previous SRM catalog listing you can rename DATA1 using the statement:

```
RENAME "*1" TO "RESULTS1"
```

In a file name completion command, a wildcard argument must match *only one* file name. Therefore, the statement:

```
RENAME "D*" TO "RESULTS1"
```

would cause the following error:

```
ERROR 295  Wildcard matches >1 item
```

Using the Escape Character

An escape character is specified when you enable WILDCARDS. The escape character cancels the special meaning of a wildcard character. The allowable values for the escape character are \ or '. The null string ("") can be specified to disable escape character processing. If an invalid escape character is specified the following error is generated:

```
ERROR 290  Invalid ESCAPE character
```

If you enabled WILDCARDS using the following statement:

```
WILDCARDS UX;ESCAPE "\"
```

you could use this statement to purge the program, `my_program*` shown in the following HFS catalog listing:

```
PURGE "my_program\*"
```

```
: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 24-Apr-90 12:01  RWXRWXRWX    18     9
my_program*    PROG     41   256 24-Apr-90 15:42  RW-RW-RW-    18     9
my_program1    PROG    412   256 24-Apr-90 14:32  RW-RW-RW-    18     9
freddy         ASCII    50   256 24-Apr-90 12:02  RW-RW-RW-    18     9
```

Restrictions on the Use of Wildcards

You can use wildcards only in the rightmost segment of a directory path. (That is, after the last "/" of the path name.) The following statements are all valid:

```
PURGE "*"
RE-SAVE "/CHARLIE/*a*b"
COPY "/PROJECTS/Project_one/KATHY/*" TO "/PROJECTS/Project_one/JOHN
```

However, the following statements are invalid, and would cause the message `ERROR 53 Improper file name` to appear:

```
PURGE "*/*"
RE-SAVE "/*/my_program"
CAT "/PROJECTS/P*/CHARLIE/DATA*"
```

General File Management Operations

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

Closed versus 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 the following statement is currently active for the file:

  ASSIGN @Io_path TO *file_name*

  Files are closed by this statement:

  ```
  ASSIGN @Io_path TO *
  ```

- Directories are closed when they are not the *current working directory.*

`MASS STORAGE IS "/USERS/MARK/MY_DIR"`	*Makes* MY_DIR *the current directory*

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). See the *HP BASIC 6.2 Language Reference* description of these commands for details.

Protecting Files

You can protect files from being read, over-written, or destroyed by other system users. Note that file protection is different for each of the three directory types.

HFS File and Directory Permissions

For HFS directories, you can use PERMIT 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.)

Nine permission bits for HFS files and directories are 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 CHOWN. (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 *HP BASIC 6.2 Language Reference* entries for CHOWN and CHGRP for further information.) With BASIC, the system owns all files and directories with an owner identifier of 18.
- GROUP—initially the group to which the owner of the file or directory belongs; however, the group identifier of individual files and directories can be changed with the CHGRP 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.

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 the contents of a file or directory (such as with OUTPUT, RE-STORE, or CREATE).
- SEARCH—an operation 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 is represented 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-`.

PERMIT is used to permit or restrict access to 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 PERMIT in the *HP BASIC 6.2 Language Reference*.

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

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

`---------` *before*
`RW-------` *after*

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 (the PERMIT parameters are the same as in the preceding example, but the *before* permission bits are different):

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

`R-XRW-RW-` *before*
`RW-RW-RW-` *after*

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

If OWNER, GROUP, or OTHER is not specified, the corresponding access-permission bits are not affected. The following statement sets permission bits for OWNER and OTHER, but leaves the bits for GROUP unchanged:

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

`R--R---W-` *before*
`RW-R--R--` *after*

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

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

`RW-RW-RW-` *before*
`RW-R--R--` *after*

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

```
PERMIT "File"
```

`RW-R--R--` *before*
`RW-RW-RW-` *after*

```
PERMIT "Directory"
```

`RW-R--R--` *before*
`RWXRWXRWX` *after*

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.

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. 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.

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.

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 access** (available to all workstations on the SRM) or **protected** access (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:

1. The file ASCII_1 has no public access capabilities; all access must include the appropriate password.
2. The file BDAT_1 has the READ capability public; anyone on the SRM system can read the file.
3. The directory MEMOS has READ and WRITE capabilities open to the public; anyone can create and purge files in the directory, and search through the directory with a statement like MASS STORAGE IS "MEMOS/SUB_DIR".

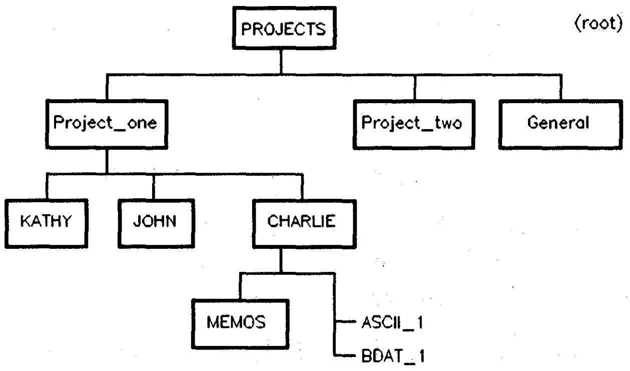
Capabilities are protected with the PROTECT statement, which associates a password with one or more access capabilities. Each file or directory can have several password/capability pairs assigned to it.

Once assigned, the password 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, if *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.

Using the directory structure in the following figure, and assuming no passwords have been assigned to the files, you could change the protections described in the list that follows.



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

   ```
   MSI "/PROJECTS/Project_one"
   PROTECT "CHARLIE",("passme":MANAGER,WRITE)
   ```

   This moves to the directory `Project_one` (immediately superior to `CHARLIE`) and executes the 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 makes CHARLIE the new working directory. 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. Everyone can read from and write to the directory, but a password is required to purge the directory or its contents. Type:

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

If you followed the steps above to protect the files and directory in CHARLIE, 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). Only the MANAGER (M) access capability on the directory MEMOS is 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. The password must be enclosed between < and > and must immediately follow the name of the file or directory it protects. For example:

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

Exclusive Access: Locking SRM and SRM/UX Files. Although allowing users to share SRM or SRM/UX files saves disk space, it introduces the danger of several users trying to access the file at the same time. To avoid problems, you can use LOCK and UNLOCK to secure files during critical operations. LOCK establishes exclusive access to a file; the file can only be accessed from the workstation at which the LOCK was executed. Typically, you LOCK all critical files, read data from files, update the data, 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. For more information, refer to the descriptions of the ASSIGN, LOCK, and UNLOCK keywords in the *HP BASIC 6.2 Language Reference*.

Locking and Unlocking SRM Files. You can get sole access to an SRM file (or an SRM/UX file) with the LOCK statement. The same file can be locked several times in succession. You must cancel each LOCK with a corresponding UNLOCK.

Using ASSIGN to re-open a locked file unlocks the file. 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, requiring a LOCK.

```
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.

LIF Protect Codes

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

To protect the file SECRET with the protect code BS, use the statement:

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

The protect code must then be specified with the file name to allow access. To RENAME the protected file SECRET, use the statement:

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

If a file has a protect code, it must be included in file specifiers for mass storage statements that *write* to that file or directory. Mass storage statements that *read* a file or directory do not require the protect code. Such statements include CAT, LOAD, LOADSUB, GET, and COPY.

To assign an I/O path name to the file named SHHHH, you 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. 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.

Renaming a file changes the file name in the directory, and leaves everything else intact, including the protect codes.

You can also assign a protect code to a BDAT file when you create it. For example:

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

To change a protect code, simply execute a new PROTECT statement:

```
PROTECT "Example<xx>","yy"     new protect code is yy
PROTECT "Example<yy>","  "     two blanks cancel protection
```

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<  >"
```

For details on the PERMIT statement, see the *HP BASIC 6.2 Language Reference* For information on SRM permissions, see *Using HP BASIC/UX 6.2.* To lock files so they can be accessed only by the person who locked the file, see the *HP BASIC Programming Guide*, "Data Storage and Retrieval" chapter, "Locking Files" section.

Copying Files

The COPY statement allows you to duplicate files. Any type of file may be copied.

COPY Statements

Example	Explanation
`COPY "ExistFile" TO "NewFile"`	Duplicate `ExistFile` into `NewFile`
`COPY "F2" TO "/users/dan/F2"`	Copy the file `F2` to another directory

To copy a file, you must have the correct permissions (described in "HFS File and Directory Permissions") that allow you to READ, WRITE, and SEARCH. You cannot copy whole directories, although you can copy a file from one directory to another. Unlike the HP-UX `cp` command, you cannot specify only a directory name to copy a file to that directory; you must include the file name.

If you are copying from an SRM or LIF disk to an HFS volume, use the `COPY` command in BASIC and not the HP-UX utilities `srmcp` or `lifcp`. Using `srmcp` or `lifcp` causes errors due to special headers on HFS files.

If the new file name (name of the copy) is the name of an existing file, you receive an error message: `ERROR 54 Duplicate file name`.

See the *HP BASIC 6.2 Language Reference* for more information on `COPY`.

Renaming Files

Renaming files allows you to change the name of a file. You will get an error message if the new name is the same as an existing file..

Rename Statements

Example	Explanation
`RENAME "MyFile" TO "YourFile"`	Renames the file `MyFile` to a file named `YourFile` in the current directory.
`RENAME "This" TO "../engineers/That"`	Renames the file `This` to a file named `That` in the `engineers` directory: actually moves the file to the new directory.
`RENAME "MyFile" TO "../MyFile"`	Moves the file `MyFile` to the parent directory

To rename a file, you must have the correct permissions that allow you to READ, WRITE, and SEARCH in the directory in which you are renaming the file.

See the *HP BASIC 6.2 Language Reference* for more information on RENAME.

Purging (Deleting) Files or Directories

Purging a file deletes the directory entry for the file.

Note Once a file is purged, there is no way of retrieving the information it contained.

The following statement removes the file "Old_stuff" from the current directory:

```
PURGE "Old_stuff"
```

If you created directories in the module on "Creating Directories", try the following examples. It's assumed you are in your home directory.

PURGE Statements

Example	Description
`PURGE "SAMPLE"`	You get an error message because the directory is not empty: it contains other directories.
`PURGE "SAMPLE/PROJECT1/FILES"`	Purge the `FILES` directory.
`PURGE "SAMPLE/PROJECT1"`	Purge the `PROJECT1` directory.
`PURGE "SAMPLE"`	Now you can purge `SAMPLE`.
`PURGE "TRIAL"`	Purge the `TRIAL` directory.

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

- To use PURGE, you must have W (write) permission on the parent directory, and X (search) permission on all superior directories.
- You cannot purge the current directory.
- Directories must be empty before you purge them (cannot contain any files or other directories).

See the *HP BASIC 6.2 Language Reference* for more information on the `PURGE` command.

Linking Files

LINK lets you link a new file name on an HFS volume to an existing file on the same volume. This command saves disk space, and lets you reference one main file with several file names.

LINK can only be used with an HFS volume. You cannot link files from one disk to another disk (however, HP-UX does support symbolic links; see the entry for cp(1) in the *HP-UX Reference*).

Using LINK

The following example links the file `COREFILE` to another file name, `NEWFILE`:

```
    LINK "COREFILE" TO "NEWFILE"
```

When you access `NEWFILE` (with an OUTPUT command, for example), the actual file accessed is `COREFILE`. In essence, LINK is much like COPY, but the actual file is not copied, and disk space is saved.

Considerations When Using LINK

The following items are important to know when using LINK:

- If the file is BDAT, ASCII, or HP-UX, an OUTPUT to the file changes its contents.
- An ENTER command on any linked file reflects any changes to the core file.
- If you RE-STORE or RE-SAVE to a file linked to other files, a new file is created and the link to the original data is broken. The RE-STOREd or RE-SAVEd file is changed, but the original data and file names referring to the original data are not changed.
- Use LINK when you want to save disk space and you want changes to be reflected in all linked files.
- Use COPY when you want two separate files, with changes reflected in only one copy of the file.

See the *HP BASIC 6.2 Language Reference* for more on LINK.

LINK is also programmable. See *HP BASIC Programming Guide* for details.

7

Editing and Storing Programs

One of the advantages of using BASIC/UX is that it makes entering, editing, and storing programs a simple task. This chapter introduces you to the concepts and skills involved in entering and storing BASIC/UX programs.

Terminology

Knowing the following terms will help you learn how to edit and store programs:

keyword	A group of characters recognized by BASIC/UX to represent some pre-defined action, such as CAT, LOAD, and 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. The maximum length of a command or program line is two CRT lines (up to 256 characters). On most models, this is 160 characters when you enter lines from the keyboard. Examples of statements are: `CAT ":,700"` `LOAD "MyProg"` `COPY "MyProg" TO "BackupFile"`
command	A statement that is executed from the keyboard.

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"
```

In general, a statement can be used as *either* a command or program line:

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

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

- A program line contains a leading line number; for example:

```
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]
```

or stored as program lines (such as DEL and SCRATCH):

```
SCRATCH A
```

The *HP BASIC 6.2 Language Reference* indicates whether or not a keyword is keyboard executable, programmable, or both.

The EDIT Mode

You can enter a program by typing program lines (line number and statement) on the normal keyboard input line of the CRT and pressing Return. But it is usually better to use EDIT mode because you can see several program lines at one time, and the BASIC editor checks the syntax of each line as you enter it.

Entering the EDIT Mode

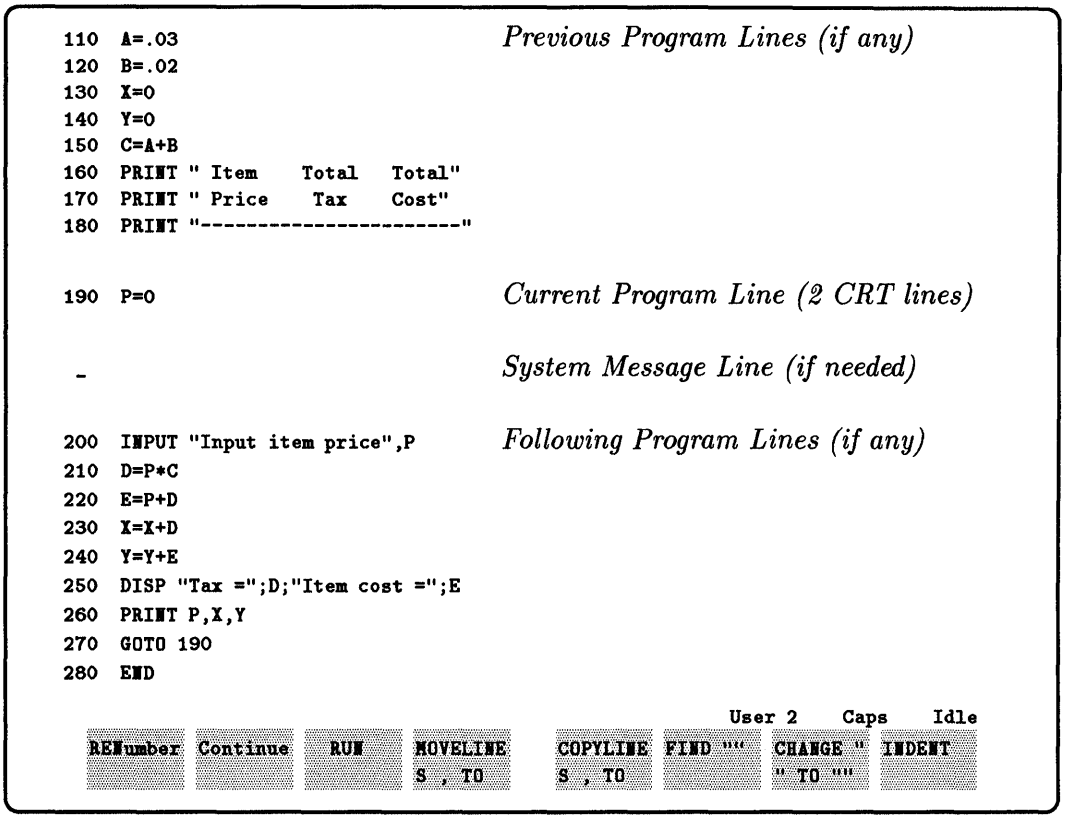
To enter the editor using an ITF keyboard, type:

1. Press (Reset) ((Shift)-(Break)) to halt any currently running program.
2. Enter EDIT mode by doing either of the following:
   a. Type:

      ```
      EDIT (Return)
      ```

   b. Press `Edit` ((f1) *in the User 1 menu*) and then press (Return)

The EDIT screen has this format:



```
110  A=.03                               Previous Program Lines (if any)
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                                 Current Program Line (2 CRT lines)

 _                                       System Message Line (if needed)

200  INPUT "Input item price",P          Following Program Lines (if any)
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
                                                          User 2   Caps   Idle
```

7

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 "Keys Used for Scrolling the Program" for details). To edit a particular line, 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, the default line number for the first program line.

The EDIT command is not programmable.

You cannot use EDIT mode while a program is running.

Parameters Allowed with EDIT Command

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 places the program on the CRT so that line 140 is in the current-line position, and new line numbers increment by 20.

```
EDIT 140,20
```

If the increment parameter is not specified, the computer assumes a value of 10. The following command places the program on the CRT so that line 1000 is in the current-line position, and new line numbers increment by 10.

```
EDIT 1000
```

When the line identifier is not supplied, the computer has different 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.

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, if you want to edit a sorting routine that begins with a line labeled `Go_sort`, type:

```
EDIT Go_Sort
```

The line labeled `Go_sort` is placed in the middle of the display, with any lines that were immediately above and below this line.

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.

Softkey Menu Changes

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.

While in EDIT mode, 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 returns you to the menu that was in effect when you entered EDIT mode.

Correcting Typing Mistakes

If you make errors while typing, use the Back space or ◀ and ▶ keys to move the cursor to the errors, then type them correctly. To insert characters, position the cursor to the right of where you want to insert, press Insert Char, and type characters. Press Insert Char again to return to the typeover mode.

Entering and Storing a Program Line

To enter a program line, type the desired characters at the keyboard. Once the line is exactly as you want it, you must press Return to store it. The cursor may be any place on the line when you store it; the system will read the entire line. For practice, type the lines shown below (the line numbers to the left are supplied for you).

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

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 BASIC stores the program line. Upon LISTing or EDITing the program, however, BASIC uses these conventions:

- Keywords are *all* upper-case letters (CAT, LOAD, DISP).
- Identifiers (variable names, line labels, or subprogram names) must begin with a capital letter and otherwise be lowercase (for example, "Word", "Label").
- Accented characters (in the ASCII code range CHR$(16) to CHR$(244)) remain as entered. These characters do not occur in keywords.

You usually do not need to use Shift when entering a line, because BASIC automatically changes all letters to the proper letter-case. However, 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). For example, "CAT" would result in a keyword conflict, but "Cat" would not.

The BASIC Editor Checks Syntax

Before storing a program line, the computer checks for syntax errors, and also changes the letter-case of keywords and identifiers. **Syntax** describes the way keywords, parameters, etc. are put together to form a legal statement.

Immediate syntax checking is a major advantage of writing programs in the BASIC editor. Many programming errors can be detected while editing, which increases the chances of a program running properly, and cuts debugging time. If the line's syntax is correct, the line is stored, and the next line number appears in front of the cursor.

If BASIC 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
```

You might not always agree with the Editor's diagnosis of the exact error or the error's location. However, an error message definitely indicates that something needs to be fixed. You can find a complete list of error messages and their meanings in the "Errors" section of the *HP BASIC 6.2 Language Reference.*

Keys Used for Editing the Current Line

To edit a line, it must be the **current line**—the line that the cursor is on. The following table gives a quick overview of the standard editing keys to use while editing the current line. The "Keyboard Information" chapter at the end of this manual lists key definitions.

Editing Features

Editing Feature	Explanation
Normal cursor (blinking underscore _)	When 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.
Insert char	Changes the cursor to the insert cursor (see below), and enters insert mode (any characters typed appear before the current cursor position, and the cursor and subsequent characters shift 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.
Delete char	Deletes the character pointed to by the cursor. Subsequent characters on the line are shifted one position to the left.
Clear line	Deletes all characters from the cursor to the end of the current line.
Shift-Clear line	Clears the entire current line.
`Set Tab` (f5) and `Clr Tab` (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.

7

Editing Features (continued)

Editing Feature	Explanation
Any Char (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, but it is not on your keyboard. Press the Any Char key. The following message appears below the current line: `ENTER 3 DIGITS, 000 THRU 255` 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 you press a key that is not part of a 3-digit number in the proper range, the ANY CHAR operation is aborted and the key performs its normal function.
Softkeys f1 thru f8 (in the User 1, 2, and 3 menus of ITF keyboards)	These keys produce characters and system-key presses, as if you had typed them at the keyboard. See the "Using Softkeys" section of the "Introduction to the System" chapter in this manual for details.

Keys Used for Scrolling the Program

All EDIT mode text-entry capabilities apply to the current line. You must move a line to the current-line position before you can edit it. 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. The text on the screen is scrolled so that you are always editing the line in a window in the middle of the screen.

Scrolling Keys

Editing Key	Explanation
▲	Scrolls the program up one line, so that you will be editing the next program line.
▼	Scrolls the program down one line, so that you will be editing the preceding program line.
Shift-▲	Jumps to the end of the program.
Shift-▼	Jumps to the beginning of the program.

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 key.

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

A new line number appears between line 90 and line 100.

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

Type and store the inserted lines in the normal manner. Appropriate line numbers will appear automatically. The insert mode is canceled by pressing the Insert line key again, or by performing an operation that causes a new current line to appear (such as scrolling).

Deleting and Recalling Lines

Lines can be deleted one at a time or in blocks. The Delete line key deletes the current line.

Before deletion:

```
90   PRINT "Line 90."
100  PRINT "Line 100."    Make this the current line, then press Delete line
110  PRINT "Line 110."
```

After deletion:

```
90   PRINT "Line 90."
110  PRINT "Line 110."    New "current line"
```

If you press Delete line by mistake, you can recover the line by pressing Recall (f8), then store it by pressing Return. BASIC/UX 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 Recall. Shift-f8 cycles through from the current line to the more recent lines. You can clear this recall buffer by executing SCRATCH R to keep others from seeing passwords that you have typed.

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.
- Delete line saves the line in the recall buffer (DEL does not).

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

Copying Lines (By Changing Line Numbers)

Although the computer supplies a line number automatically, you can change that line number easily. Use Backspace to place the cursor over the line number and type the line number you want to use. You can copy existing lines to another part of the program by changing their line numbers. However, there is an easier way to copy program lines—by using the COPYLINES command, described in "Copying Program Segments" in the section on "Global Editing Operations."

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

- Changing the line number of an existing line causes a *copy*, not a move.
- Entering a line with the same number as an existing line *replaces* the existing line.

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.

Summary of Global Editing Operations

BASIC Command	Purpose and Example Command	Softkey User 2 Menu
REN	Renumbers the program (or a specified segment of the program). `REN 100,10`	f1
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
FIND	Searches the program for a specific textual pattern. `FIND "a pattern"`	f6
CHANGE	Searches for a textual pattern, but allows you to optionally change it to a new pattern. `CHANGE "old text" TO "NEW CHARACTERS"`	f7
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 a default softkey.

Renumbering a Program

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

Commands to Renumber Program Lines

Command	Explanation
`REN`	Renumber the program lines using 10 as the first line, and increment lines by 10 (default).
`REN 100,5`	Renumber the program using 100 as the first line, number and increment lines by 5.
`REN 100,5 IN 100,200`	Renumber only in the line range of 100 to 200; make the first line of this segment 100, and increment by 5.

Indenting a Program

INDENT is a non-programmable command. You can use it to scan an entire program, indenting it to nest program segments defined by the following cases:

- 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).

Commands to Indent Program Lines

Command	Explanation
`INDENT`	Indent lines to show nesting of branching constructs (FOR ... NEXT, REPEAT ... UNTIL, etc.).
`INDENT 7,2`	Place the first character of the outermost construct in column 7 and space each indentation by 2 characters (INDENT 6,2 is the default).
`INDENT 7,0`	Remove all indentation.

The following program shows the indentation performed by this command:

```
INDENT 7,2
```

Notice how indentation improves readability.

```
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
```

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. 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. Line beginnings drop back to the left when the nesting level decreases to one increment less than 72..

Finding Textual Patterns

When programs are larger than a couple of screenfuls, the computer can search for items such as variable names, numeric or string literals, and comments. Use the non-programmable FIND command to do this.

The following example searches the current program for the letters `A pattern`, beginning at the current line. These letters may be a variable name, a string or numeric literal, a comment, or a portion of any of these.

```
FIND "A pattern" (Return)
```

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)
```

When you execute this command, BASIC begins a search for these characters. The following message is shown in the message/results line below the keyboard input line.

```
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"
```

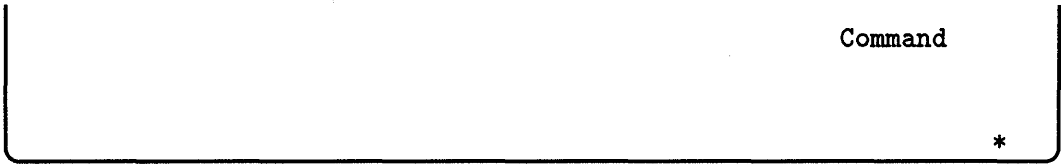
You can choose any of the following actions:

- Edit the line (optional): move the cursor to change, add, or delete characters.
- Press (Return) to store the edited (or unchanged) line.
- Scroll the program up or down (with (▲) or (▼)) to cancel 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 the line's syntax, FIND searches 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, FIND 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. To remind you that you are in this mode BASIC displays these prompts at the bottom, right-hand corner of the screen:



```
                                              Command

                                                    *
```

If you want to abort the FIND command, then use the Break (Clr I/O) key to cancel the mode. BASIC/UX 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 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
```

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

```
Finding "Old text"
```

CHANGE pauses when it finds the first occurrence of the search pattern; however, CHANGE also replaces the old pattern with the new one, and waits for you to confirm or reject the change:

```
200   PRINT "New characters."

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

- You can edit the line first. To confirm the change, press Return.
- To reject the change, press Continue (f2).

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 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

COPYLINES provides an easy way to duplicate several lines of BASIC code in another location of the program. You must make sure the destination line numbers do *not* already exist.

The following example 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 existing line number.*

Moving Program Segments

Use MOVELINES to move several program lines at a time. You must make sure the destination line number does *not* already exist.

The following command moves lines 32 through 127, inclusive, to a spot beginning at line 453:

```
MOVELINES 32,127 TO 453  [Return]
```

The following command 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 existing line number.

You may frequently use MOVELINES to move program lines from a main program into a separate subprogram (defined by SUB and SUBEND). 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 you type 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

DEL can be used to delete several lines in a single operation. Delete blocks of program lines 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
```

Subprograms or user-defined functions in your program 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 *HP BASIC Programming Guide.*

DEL is not programmable and cannot be used while a program is running.

Making Programs Readable

Good 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.

BASIC/UX 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.

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/UX language to make the program more easily understood. Which version would you rather support?

```
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
```

Securing Program Lines

There may be times when you want to keep portions of your programs from being read or used by others. With BASIC, you can prevent others from reading or executing a program unless you give the authorization. SECURE prevents program line(s) from being listed.

Note Once a program is secured, it *cannot* be unsecured. Therefore, you should keep an unsecured backup copy of all programs.

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      .
        .
        .
```

7

If you want the whole program to be secured, use this statement:

```
SECURE
```

BASIC/UX also provides a method to prevent unauthorized execution of programs. You can write a routine that includes the following statement; it 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 execute.

`SYSTEM$("SERIAL NUMBER")`	*Reads the computer's ID PROM or returns encoded ID Module contents*

In order to decode an ID Module's contents, use the "ID_MODULE" program supplied on the *Manual Examples* disk.

Exiting EDIT Mode

You can terminate the EDIT mode in many ways. Your choice depends upon what you want to do next. If you simply want to return the CRT to its normal state, press Stop (PAUSE) or Clear display.

You can also leave EDIT mode by proceeding with another operation. The most useful choices in this case are LIST, CAT, Reset (Shift-Break), RUN (f3), or Step (f1, system menu). EDIT mode is also terminated by a GET or LOAD, and by any operation that uses the display. Some examples follow:

- List the program on the current PRINTER IS device (usually the CRT) by executing the following command.

  ```
  LIST Return
  ```

- You can store a program currently in memory in a file named `MyProg` on the default volume by executing the following statement:

  ```
  STORE "MyProg" Return
  ```

- You can run a program currently in memory by executing the following statement:

  ```
  RUN Return
  ```

  or pressing RUN.

Storing the Program on Mass Storage

To write a program to a mass storage device, use either SAVE or STORE. 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 (nearly five times faster than files stored with SAVE).
- SAVE creates an HP-UX type file when saving on an HFS volume.
- SAVE creates ASCII type files when saving on LIF or SRM volumes.

  The main advantage of an ASCII file is that it can be read as data by a BASIC/UX program or by LIF-compatible devices (such as other HP computers and terminals). LIF is the acronym for Logical Interchange Format, a disk 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.)

  For information about converting ASCII type files to HP-UX type files, see "Making ASCII Type Files HP-UX Type Files" in the chapter "Using HP-UX Commands in BASIC/UX."

Using STORE

The following command creates a program file called `MyProgFile` on the current default volume:

```
STORE "MyProgFile"
```

If you get error 54, it means a file with the name you are using is already on the disk. 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.
- Replace the existing file with a newer version. To replace an existing file, use RE-STORE. The command to replace a program file called `BEAMS` is:

  `RE-STORE "BEAMS"`   `RE-STORE` *must include the hyphen*

- PURGE the old file, then STORE the new one.

Using SAVE

The SAVE (or RE-SAVE) keywords, store information as ASCII characters, and you must use GET to retrieve these programs.

- Using SAVE on HFS or SRM/UX volumes creates HP-UX type files
- Using SAVE on SRM or LIF volumes creates ASCII type files.

SAVE 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.

`SAVE "WHALES"`	*Saves the entire editor contents*
`SAVE "LastPart",500`	*Saves from line 500 to end*
`SAVE "Sorter",Sort,Printout`	*Saves lines from label Sort to label Printout*
`RE-SAVE "Analysis"`	*Replaces old Analysis program with new version*

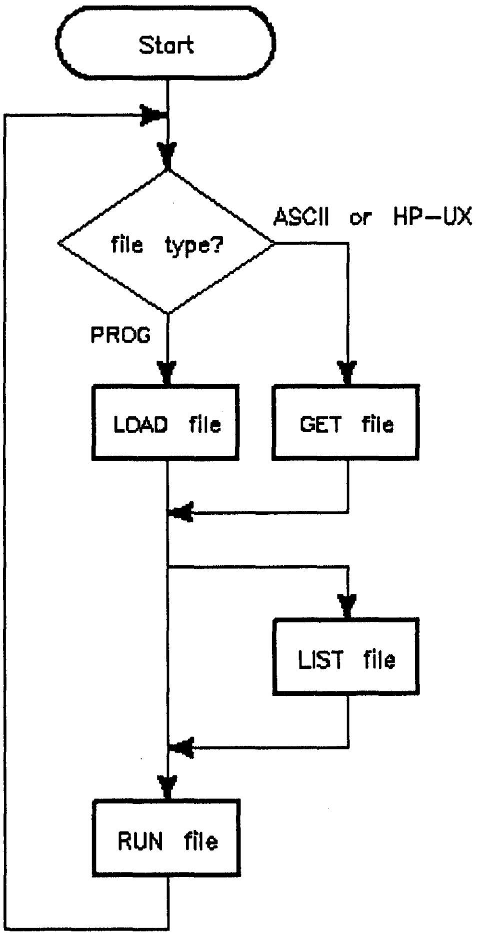
Note Use SAVE to save to a new file that doesn't exist. Use RE-SAVE if the name already exists. Use the GET statement to retrieve a file that has been SAVEd.

8

Loading and Running Programs

Since two different types of files can be created for BASIC/UX, there are different commands for each file type:

- Programs STORE'd (or RE-STORE'd to an existing file) create file type PROG, and you must `LOAD` the program into memory and then run it.
- Programs SAVE'd (or RE-SAVE'd) are stored as ASCII characters, and you must `GET` the program and then run the program.
- Programs SAVE'd on HFS volumes are HP-UX type files.
- Programs SAVE'd on SRM or LIF volumes are ASCII type files.



Loading and Running Files

Loading Programs

To load a program, follow this procedure:

1. Use CAT to locate your program. If necessary, include the volume specifier or the directory path as described in "Files, Directories, and Volumes."

   Your listing should look similar to the example below:

```
: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
```

2. Load the program into computer memory.

   Use LOAD for program files designated as PROG under TYPE on the CAT listing.

```
LOAD "MY_PROG"
```
   File type is PROG

   Use GET for program files designated as ASCII or HP-UX on the CAT listing.

```
GET "VISI_TOOL"
```
   File type is ASCII or HP-UX

   To load a program that is not located on the default drive, use the volume specifier.

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

LOAD and GET can be executed from the keyboard as commands or included in a program. When executed as commands, they bring a program into the computer's memory so it can be edited or run. When included within a program, they 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 before loading a new file. For example:

`LOAD "CANNON"`	Loads program into memory without running it
`LOAD "CANNON",10`	Loads program into memory and starts running it from line 10
`LOAD "CANNON",Here`	Loads program into memory and starts running it from line labeled `Here`

If you specify a line label or number, it must identify a line in the main program segment (not in a subprogram or user-defined function). See the "Program Flow" chapter of the *HP BASIC Programming Guide* for further information.

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

If the file is not a PROG file, LOAD is not performed and Error 58 (improper file type) is reported.

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 load an ASCII or HP-UX file, use GET followed by the file name. The following statement clears any BASIC/UX program currently in memory, and loads an ASCII or HP-UX file called FORMULA, assuming the file contains valid program lines.

```
   GET "FORMULA"
```

If the first line does not start with a valid line number, GET is not performed and error 68 (syntax error during GET) is reported.

The lines are entered into program memory if:

- the line contains valid program lines that were placed in the file by a SAVE operation
- line numbers are still valid after any renumbering that is specified.

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 different version of BASIC/UX from the one used for the GET operation.

If the file is not an ASCII or HP-UX file, GET is not performed and Error 58 (improper file type) is reported.

Using GET to Append and Run

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.

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

```
GET "RATES",100,10
```

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

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

deletes any existing lines from 250 to the end of the program in memory, renumbers the contents of file RATES, and appends it to the program in memory beginning at line 250. Program execution begins at line 100. Any combination of line identifiers is allowed, but the line specified as the start of execution must be in the main program segment (not in a SUB or user-defined function). Execution does not begin if there is 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 the *HP BASIC Programming Guide*.

Running a Program

To run a program you just loaded, type:

```
RUN Return
```

On ITF keyboards, f3 (RUN)in the System menu, and User 1 and 2 menus serve the same purpose. If key labels are not currently displayed, execute KEY LABELS ON or press Menu to turn them on.

Note Some software is secured against being run without proper authorization. This is usually accomplished by requiring a special codeword 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 need to get it from your system administrator.

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

`RUN 200`	*Begin execution at line number 200*
`RUN` *Line_id*	*Begin execution at the line label*

Prerun

BASIC/UX automatically performs a **prerun** when you execute RUN or press `RUN`. Prerun completes the following tasks:

- *Reserves 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.
- *Locates all the context boundaries.* These are defined by the END, SUB, SUBEND, DEF FN, and FNEND statements.
- *Ensures correct interaction between lines.* Although BASIC/UX checks for syntax errors before it stores a program line, some errors can't be detected by looking at a single line. An example of this kind of error would be an improper matching of statements like FOR ... NEXT and IF ... END IF. At prerun, BASIC/UX also links line identifiers and line numbers with all references to them, to make program execution faster.

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. **Live keyboard** means commands can be executed during a running program. A principal use for live keyboard commands is troubleshooting and debugging programs, as discussed in the "Debugging Programs" chapter of the *HP BASIC Programming Guide*. See "Introduction to the System" in this manual for tables showing how to pause, stop, and continue a program.

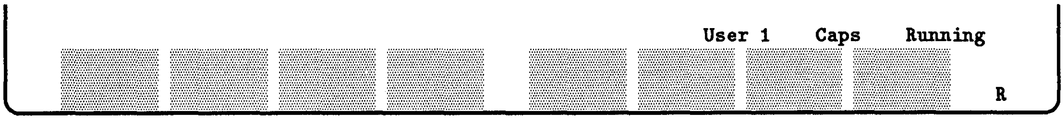
Controlling Program Execution

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
```

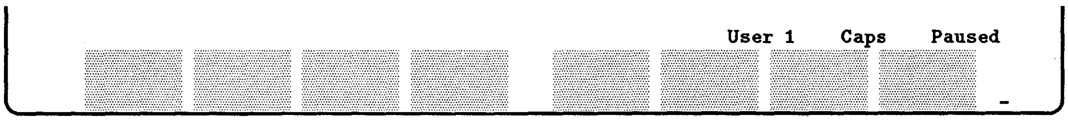
1. After you enter the program, execute RUN and observe the CRT. The DISP message appears in the display line, the printout area fills with a sequence of numbers, and the run light indicates that a program is running.

Program-Status	**Running**
Run Light	▨



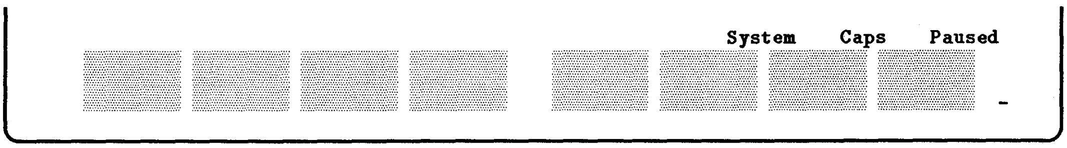
2. Press Pause (Stop). The printout of numbers stops, and all the data on the CRT remains unchanged. The run light now indicates that the program is paused and can be continued. The program line that appears at the bottom of the CRT is the next line that will be executed when program execution resumes.

Program-Status	**Paused**
Run Light	_



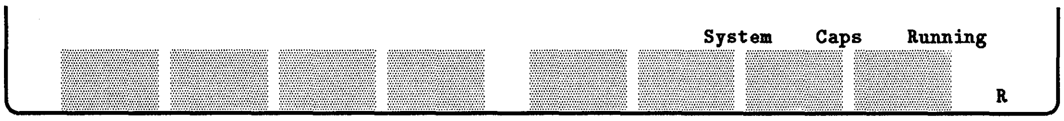
3. Press `Step` (f1) a few times. The program is executed one line at a time, as indicated by the lines changing at the bottom of the CRT. The program is still paused and continuable after each press of the STEP key. The STEP key can be a great help when you are trying to find certain kinds of problems.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

Program-Status	**Paused**
Run Light	_



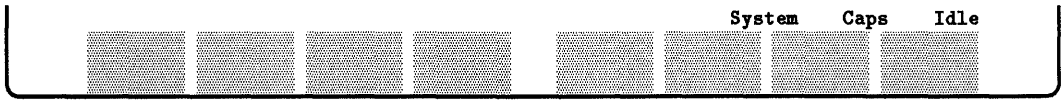
4. Press `Continue` ((f2)). The printout on the CRT resumes with the next number in the sequence. The run light again indicates that a program is running.

Program-Status	**Running**
Run Light	▨



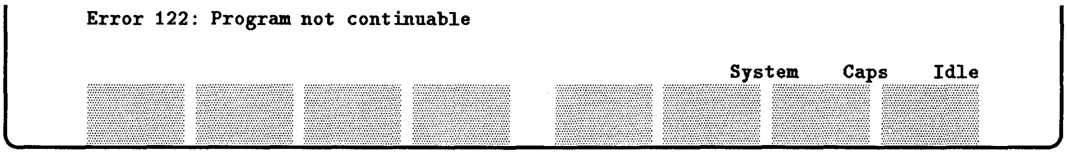
5. Press Stop ((Shift)-(Stop)) The printout of numbers stops, and all the data on the CRT remains unchanged. However, the run light is off, indicating a stopped condition.

Program-Status	**Idle**
Run Light	*(blank)*



6. Press `Continue` (f2). An error results, because a stopped program cannot be continued.

Program-Status	`Idle`
Run Light	*(blank)*



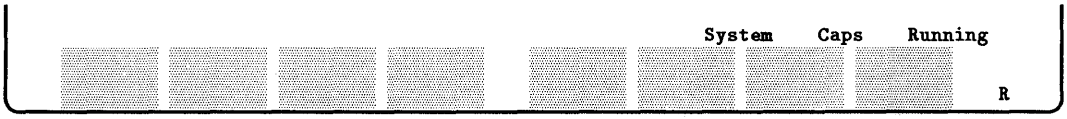
```
Error 122: Program not continuable

                                          System     Caps     Idle
```

7. Press `RUN` (f3).

   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.

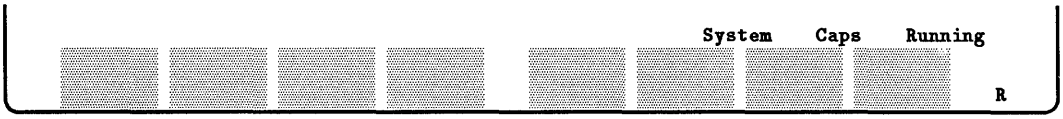
Program-Status	`Running`
Run Light	▨



```
                                      System     Caps     Running
                                                                    R
```

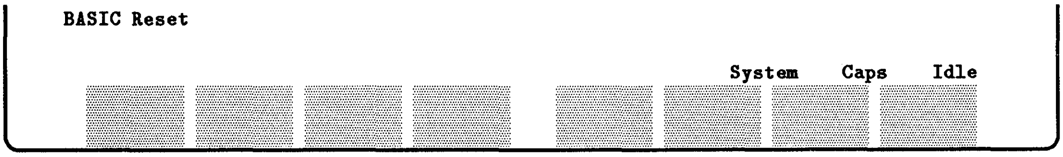
8. Type `X=1` and press (Return). Notice that the numbers being printed start over with "1". The live keyboard was used to change the value of "X", and the program used the new value from the keyboard.

Program-Status	`Running`
Run Light	▨



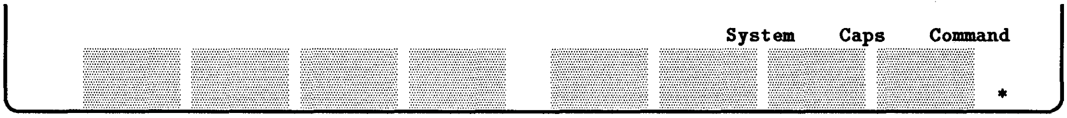
9. Press Reset ((Shift)-(Break)). The program stops and the data remains in the printout area, but the display line is cleared and the message `BASIC Reset` appears at the bottom of the CRT. Although the clearing of the display line seems like a minor effect, it indicates an important point. Reset and Stop have different effects on interfaces and peripheral devices. This aspect of Reset is summarized in the "Reset Tables" in the "Useful Tables" section of the *HP BASIC 6.2 Language Reference* and is discussed fully in the *HP BASIC 6.2 Interface Reference*.

Program-Status	`Idle`
Run Light	*(blank)*


BASIC Reset

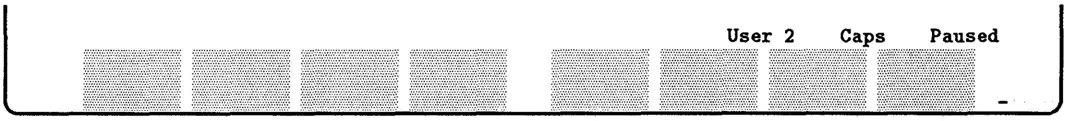
10. Press RUN (f3). Then type `WAIT 5` and press Return. The run light changes to indicate that a keyboard command is being executed and the printout is delayed for five seconds while the live keyboard command is processed. Actually, the run light changed when the `X=1` command was executed in step 8,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

Program-Status	Command
Run Light	*



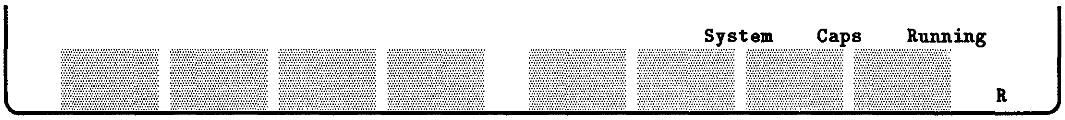
11. Press Pause (Stop) and then type `EDIT` and press Return. The display on the CRT changes to show the program. The line you were editing last appears in the current-line position. The run light is still visible in the lower right-hand corner and it indicates that the program is paused.

Program-Status	Paused
Run Light	_



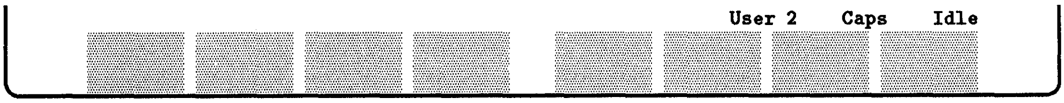
12. Press CONTINUE (f2). The CRT returns to normal mode, and the printout of numbers continues in sequence. However, the previous data on the display was lost when the CRT was used for EDIT mode.

Program-Status	**Running**
Run Light	▨



13. Press Pause (Stop). 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. 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.

Program-Status	`Idle`
Run Light	*(blank)*



9

Using HP-UX Commands in BASIC/UX

The EXECUTE command is used to run HP-UX commands while in BASIC/UX. This chapter also discusses how to localize BASIC/UX error messages in other languages.

Using the EXECUTE Command

EXECUTE runs an HP-UX command in a "sub-shell". Any HP-UX variables you create while the EXECUTE command is running are erased when control is returned to BASIC/UX.

In the X Window System, the output from EXECUTE is displayed in the HP-UX (root) window that initiated BASIC/UX (if you boot directly into BASIC/UX, the HP-UX window may be "underneath" your BASIC/UX window).

Examples of the EXECUTE Command

The following table shows some examples of the EXECUTE command. For a detailed look at EXECUTE and its options, see the *HP BASIC 6.2 Language Reference*.

Example EXECUTE Statements

Example	Description	Example Results
`EXECUTE`	Run HP-UX in the window from which BASIC/UX started. Type `exit` (Return) to return to BASIC/UX.	`$`
`EXECUTE "date"`	Runs the HP-UX command `date`. Type `date` in lower-case.	`Tue Apr  5 09:55:53 MDT 1988`
`EXECUTE "more myfile"`	Run the HP-UX command `more` to show the contents of `myfile`.	`This is not a PROG file. It could have been created in HP-UX.`

How to Run HP-UX Commands in the Background

If you have an HP-UX command that takes a long time to execute or is a concurrent process (runs all the time), you can place it in the "background" so it runs at the same time you use BASIC/UX.

To run an HP-UX command in the background, add the ampersand (&&) character after the HP-UX command in the BASIC/UX statement. For example,

```
   EXECUTE "ps &"
```

executes the HP-UX command **ps** and lets you continue executing BASIC/UX commands. If you omit the &&, you cannot execute any more commands until the process is completed. To halt the process without the &, use one of the following:

- (Break)
- (CTRL)-(c)
- `exit` (Return) *if in an HP-UX shell requiring input*

If you are in the X Window System and the process continues, move the pointer to the BASIC/UX window and press Reset (Shift-Break). Use this as a last option as it may clutter the system with parts of the killed process.

How EXECUTE Displays

When you are in the X Window System, the EXECUTE command runs in the window that started BASIC/UX (root window).

If you are not in X, the screen is cleared and the EXECUTE command output is shown on a new screen.

EXECUTE When Not in X Windows

If you want to ...	Do this ...
Return to BASIC/UX	When the process completes, PRESS ANY KEY TO CONTINUE: appears. Press a key, to clear the screen and return to BASIC/UX.
Stop a process while it is running	Press Break.
Leave an HP-UX shell	Type **exit** Return.

If you are in X and the process does not stop after using the above methods, move the pointer to the BASIC/UX window and press Reset (Shift-Break).

EXECUTE Runs as a Child Process

When you run an EXECUTE command, it is processed as a **child** process. In other words, the command defines its variables only for the time the command is being processed. Those variables are lost when you return to BASIC/UX.

See *HP BASIC 6.2 Language Reference* for other options of EXECUTE, for the SAVE ALPHA OFF option which saves graphic output, and the RETURN option which allows you to return to BASIC without pausing.

See *A Beginner's Guide to HP-UX* for general information on operating in HP-UX. For details on HP-UX commands, see the *HP-UX Reference*.

Using Some HP-UX Commands and Utilities

Commands in the following table can be used in the EXECUTE statement. Refer to *A Beginner's Guide to HP-UX* for detailed information or a complete list of beginning HP-UX commands.

Some Useful HP-UX Commands

Command	Description
`cal`	Display a calendar for the current month.
`date`	Display the current date and time.
`hostname`	Display your system's "node" name.
`mailx` *user*	Send an electronic mail message to *user*.
`man` *command*	Display on-line information on *command*.
`whoami`	Display your user name.
`cp` *file newfile*	Copy *file* to *newfile*.
`ls`	List (catalog) files in current directory.
`mkdir` *dir*	Create a directory named *dir*.
`more` *file*	See the contents of *file*. Press the spacebar to continue scrolling when content list pauses.
`mv` *file newfile*	Change the name of *file* to *newfile* (`mv` can also be used to move a file to a specified directory).
`pwd`	Display the path name of the current directory.
`rm` *file*	Destroy *file* (file is irretrievable).
`rmdir` *dir*	Destroy the directory *dir* (cannot contain files).
`ps -ef`	Display all current processes running on the system.

For example:

```
EXECUTE "date"
```
displays the current date and time

Using BASIC/UX Program Files with HP-UX Commands (and Vice Versa)

Only HP-UX type files can be used with HP-UX commands and utilities. A program can exist on disk as a PROG file, an ASCII file, or as an HP-UX file:

- If you use the STORE or RE-STORE command to store a program on disk, it will be a PROG type file regardless of the file system. (You can use the LOAD command to load the program into memory.)
- If you use the SAVE or RE-SAVE command to save a program on disk in a LIF or SRM volume, it will be an ASCII type file. (You can use the GET command to load the program into memory.)
- If you use the SAVE or RE-SAVE command to save a program on disk in an HFS or SRM/UX volume, it will be an HP-UX type file. (You can use the GET command to load the program into memory.)

Saving Programs as HP-UX Type Files

In order to use HP-UX commands on a program file, you will have to SAVE or RE-SAVE the program on an HFS or SRM/UX volume. Let's look at some examples.

First, suppose that you have a program stored as a PROG file named "test_1" in the HFS directory "/users/jim". The first step is to load the program into memory:

```
MSI "/users/jim"
LOAD "test_1"
```

Now save the program as an HP-UX file in the same HFS directory with a different name:

```
SAVE "test_1.ux"
```

You can now use HP-UX commands to manipulate the HP-UX file "test_1.ux".

Note If you wish to use the file with an HP-UX command (for example, the `vi` editor), you'll need to exit BASIC/UX (or create a sub-shell with EXECUTE) first.

For our second example, suppose that you have a program saved on a LIF volume at ":,700" as an ASCII file, "test_2".

Use the GET command to load the program into memory:

```
   GET "test_2:,700"
```

Assuming the current MSI is still "/users/jim", an HFS directory, just SAVE the program as follows:

```
   SAVE "test_2"
```

On the HFS volume, the file "test_2" will be of the HP-UX type, but the original "test_2" on the LIF volume will still be an ASCII file.

Using HP-UX Files in BASIC/UX

HP-UX files (created in an HP-UX editor, for example) can be used by BASIC/UX. Use GET to bring the file into memory.

If you attempt to GET an HP-UX file that does not contain a BASIC program, you may receive the following error message:

```
   ERROR 68 Syntax error occurred during GET.
```

You cannot use the BASIC/UX editor for an HP-UX file that does not contain a BASIC program.

Converting Error Messages to Another Language

HP-UX Native Language Support (NLS) tools let you localize BASIC/UX error messages to languages other than English.

- You must be logged in as `root` or be super-user (`su`) to modify the message file. See your system administrator to do this task.
- The fileset `NLS_CORE` from partition `NLS` must be installed on your system. To check, type:

  ```
  lsf /etc/filesets | grep NLS_CORE (Return)
  ```

  If you see `NLS_CORE` returned, it means you have the fileset loaded. If you see nothing but a new prompt, you must load the fileset. See the chapter "Updating HP-UX" in the *HP-UX System Administration Tasks* manual for your series of computer. You must be system administrator to do this task.
- You should know the language to which you are converting the messages.
- You should be familiar with C language `printf` statements.
- You must be familiar with an HP-UX editor (such as `vi`).

To convert BASIC/UX error messages, use the following procedure:

1. Login as `root` or become super-user (`su`).
2. Copy a default file to `/tmp` as a working file:

   ```
   cp /usr/lib/rmb/newconfig/rmb.msgs /tmp/rmb.msgs
   ```

3. Change directories: `cd /tmp`
4. Edit `rmb.msgs` (for example, with `vi`) and change the English strings to the appropriate language equivalents.
   - Do not change lines beginning with $
   - Do not change or remove any numbers.
   - Do not remove any of the format information (for example, quoted strings are C programming language print control statements).

   `rmb.msgs` is a large file, so you may want to print the file first.

5. Execute the HP-UX `gencat` utility:

   ```
   gencat rmb.cat rmb.msgs
   ```

6. List the `/usr/lib/nls` directory to choose the language catalog where you store the new message file:

   ```
   ls /usr/lib/nls
   ```

7. Move the new file to the appropriate directory in `/usr/lib/nls`:

   `mv rmb.cat /usr/lib/nls/`*language*

   For example, if you are converting to Spanish,

   ```
   mv rmb.cat /usr/lib/nls/spanish
   ```

Be sure to include spaces between the file names in the above commands. If you receive a message like

```
mv [-f] f1 f2
mv [-f] f1 ... fn d1
mv [-f] d1 d2
```

while running the `mv` command, it means you forgot the spaces.

For more information see:

- *HP-UX Reference* for `gencat(1)`.
- *HP-UX Concepts and Tutorials: Device I/O and User Interfacing*, "Native Language Support" section.

10

Creating Environment and Autostart Files

An environment file sets system variables when you start BASIC/UX. An autostart file then is automatically run with commands you specify.

Customizing Your BASIC/UX Session

When you start a BASIC/UX session, the system looks for a file:

```
/usr/lib/rmb/rmbrc
```

to set the default environment. The environment consists of system variables that affect how the system performs some tasks.

A template environment file:

```
/usr/lib/rmb/newconfig/rmbrc
```

is available for you to customize, move to your home directory, and call `.rmbrc` Type the file name lower-case, and precede with a period (.).

To copy the default file to `/users/leslie` type:

```
COPY "/usr/lib/rmb/newconfig/rmbrc" TO "/users/leslie/.rmbrc"
```

Then EDIT the file and make any necessary changes based on the descriptions that follow. See "How to Create Your Evironment Files" for details.

What Variables Can Be In The Environment File?

More details about these variables follow the table.

Global Environment Variables

Name of Variable	Range of Values	Default Values	Description
`autostart`	*pathname*	n/a	Pathname of an autostart file.
`errormode`	`on`, `off`	`on`	Generate error messages for BASIC Workstation, BASIC/UX compatibility.
`graphics_buffer`	`on`, `off`	`on`	Turn on graphics buffering to speed up graphics processing time.
`heap_prealloc`	0 to space available	n/a	Preallocate heap space.
`hfs_buffer`	`on`, `off`	`on`	Turn on HFS file system buffering.
`plock`	`all`, `t`, `d`, `w`	n/a	Lock text area, data area, workspace or all of BASIC/UX into memory (any combination of `t`, `d`, and `w` is valid).
`term_control`	`on`, `off`	off	Provide access to the special terminal keyboard mappings.
`workspace`	64k to amount of RAM	1m	Size of BASIC/UX workspace (integer values only).

Here is more detail on the environment variables for BASIC/UX, as well as some additional statements you can add to rmbrc.

All examples below show an arbitrary line number created using EDIT mode.

Running an Autostart Program (autostart)

If you want an autostart program, you can specify the file with autostart. Using the program, /users/leslie/AUTOST, you would enter:

```
   50 !autostart=/users/leslie/AUTOST
```

into the .rmbrc file.

See the subsequent section, "Creating an AUTOST File."

Generate Compatibility Error Messages (errormode)

If you port programs created on BASIC Workstation systems, you may have some errors. For example, some commands (such as LOAD BIN) are not supported on BASIC/UX.

- 60 !errormode=on *error messages are generated*
- 60 !errormode=off *does not print error messages*

Graphics Buffering (graphics_buffer)

When using graphics, you can choose to have graphics buffering:

70 !graphics_buffer=on	The image is faster than when off, but it could be choppy when an image moves on the screen.
70 !graphics_buffer=off	The image is slower than when on, but it is smoother when the image moves on the screen.

Increasing the Heap Space (heap_prealloc)

To increase your heap space, type:

```
75 !heap_prealloc= additional_heap_space
```

If the *additional_heap_space* given in bytes is zero (the default value), then no additional heap space is allocated; however, if it is greater than zero, the amount of heap space specified is preallocated.

The heap-consuming BASIC operations are listed below, as well as suggested amounts of heap space to add for each one if the need arises:

Heap-consuming BASIC/UX Operations

Operation	Heap Space Required
CREATE WINDOW	17K for the default window and buffer sizes
GLOAD/GSTORE	*width* × *height* of "from" device (if given) or "PLOTTER IS" device (if no parameters)
INITIALIZE *memory volume*	*number of sectors* × 256 bytes
CSUBS	size of stored CSUB file
BPLOT	*width* × *height* for given parameters (plus size of stored CSUB)
GDUMP_R	*width* × *height* of "from" device (plus size of stored CSUB)
DUMP GRAPHICS	*width* × *height* of "from" device or `PLOTTER IS` device if no parameter is given
mass memory	operations on HFS directories will at most use 20 Kbytes
opening SRM file	each open file uses 48 bytes

HFS File System Buffering (hfs_buffer)

This variable determines how data is written to a disk:

`80 !hfs_buffer=on`	Saves data in a buffer and writes to a disk periodically. Makes system operations faster, but a greater amount of data could be lost if a power failure occurs, or the system is not properly shut down.
`80 !hfs_buffer=off`	Writes data to the buffer, then immediately to the disk. OUTPUT performs much slower.

Locking BASIC/UX in Memory (plock)

Note that `plock` disables swapping. To lock the text area, data area, workspace, or all of BASIC/UX into memory:

`90 !plock=all`	locks BASIC/UX into memory.
`90 !plock=t`	locks text area into memory.
`90 !plock=d`	locks data area into memory.
`90 !plock=w`	locks workspace into memory.

Any combination of `t`, `d`, or `w` is valid (for example, `plock=td`).

Setting Special Terminal Keyboard Mappings (term_control)

To set the terminal keyboard mapping mode, type:

`115 !term_control=on`	Accesses the special terminal keyboard mappings, such as CTRL-R (Reset) and CTRL-L (Recall).
`115! term_control=off`	Turns off the special terminal keyboard mappings.

Setting Size of BASIC/UX Workspace (workspace)

To set the size of BASIC/UX workspace use:

```
120 !workspace=2m
```

Make the value dependent on the size of programs to be run. A program with lots of subprograms (CSUBs, for example) needs more workspace than a small program. If you have difficulty increasing your workspace larger than 3 megabytes, read the chapter "Maintaining the BASIC/UX System" found in the *Installing and Maintaining HP BASIC/UX 6.2* manual.

Setting Up Automatic Device File Locking and Mapping

Set up automatic locking, memory mapping, or set `io_burst` on an I/O interface. Note that `autolock` + `automap` is equal to `autoburst`.

Determine the select code of the interface, and include this in the `.rmbrc` file:

`interface` *select_code*; *option*

where *option* is one of the following:

- `autolock`
- `automap`
- `autoburst`
- `normal.`

Mapping BASIC Mass Storage Volume Specifiers to HFS Directories

If you have programs that access mass storage devices with the volume specifiers, you can map a volume specifier to an HFS directory with this entry:

`disk` *scba,volume,unit* = *directory name*

where elements represent the following:

scba	Select code * 100 + bus address
,volume	The volume number (optional; use the comma if you include this)

,unit	The unit number (optional; use the comma if you include this)
directory name	An HFS directory. This could be the directory under which the disk is mounted.

To map a device on select code 7, bus address 2, volume 1 mounted under `/disk1`.

```
disk 702,1 = /disk1
```

How to Create Your Environment File

To make the `.rmbrc` file compatible with HP-UX, you must either create a new `.rmbrc` file and place it in your HOME directory, or copy the system default and modify it. To format the file, use:

- an HP-UX editor (`vi`, for example)
- the BASIC/UX editor.

Using HP-UX Editor

1. Be in HP-UX (`QUIT` or `EXECUTE` from BASIC/UX).
2. Copy the default `rmbrc` file:

   ```
   cp /usr/lib/rmb/newconfig/rmbrc $HOME/.rmbrc
   ```

3. Modify the `.rmbrc` file, for example: `vi .rmbrc`

Precede comments with the # character; *include statements without spaces between variables and values*. For example:

```
# This is a sample rmbrc file edited in HP-UX
interface 7;autolock
errormode=off
workspace=1m
# End of sample rmbrc file
```

Files created with an HP-UX editor cannot be modified using the BASIC/UX editor.

Using the BASIC Editor

1. Enter BASIC/UX: `rmb`
2. Make sure you are in your HOME directory (`EXECUTE "echo $HOME"` tells you your home directory).
3. Copy the default file:

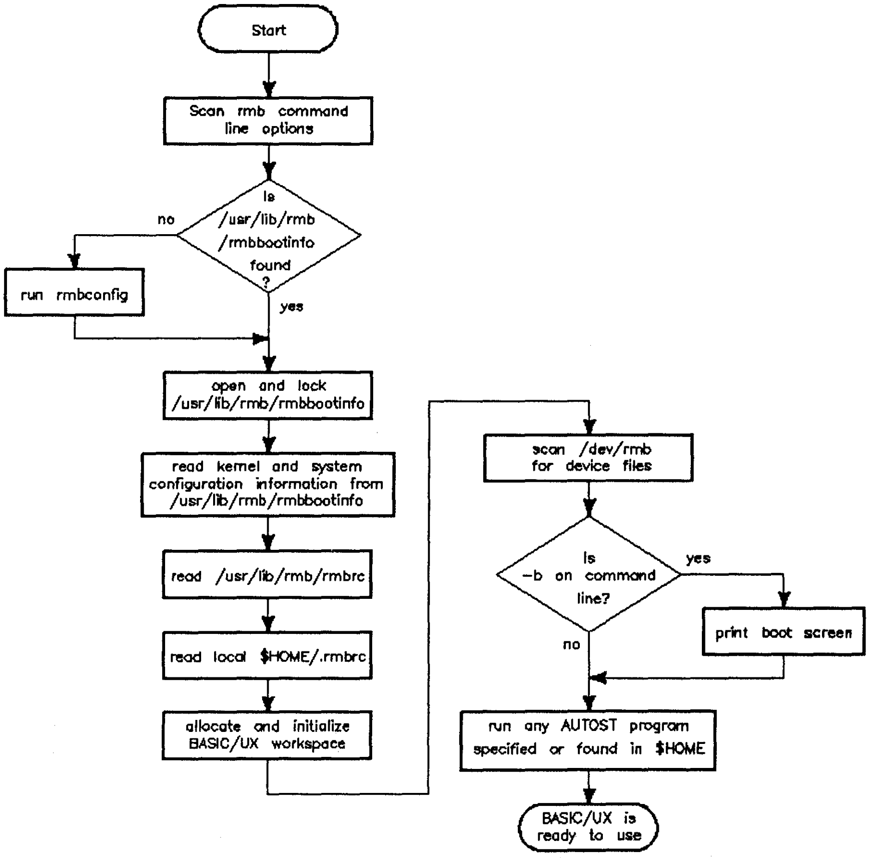
   ```
   COPY "/usr/lib/rmb/newconfig/rmbrc" TO ".rmbrc"
   ```
4. Edit the new file: `EDIT` and make modifications (see below for an example file).
5. Save the file: Shift-Clear line `RE-SAVE ".rmbrc"`

All lines must be preceded with line numbers and an exclamation mark. Precede comments with `!#` or `REM`. For example:

```
10 REM This is a sample rmbrc file edited in BASIC
20 !interface 7;autolock
30 !errormode=off
40 !workspace=1m
50 !# End of sample rmbrc file
```

Creating an AUTOST File

If you always want to perform the same task each time you enter BASIC/UX (for example, you always want to run the same program), you can use an AUTOST (autostart) program file. The file must be of the "PROG" type and must have the file name "AUTOST". The AUTOST program will be found and will run at the end of the boot process, which is shown below:



Start
Scan rmb command line options
Is /usr/lib/rmb /rmbbootinfo found ?
no
yes
run rmbconfig
open and lock /usr/lib/rmb/rmbbootinfo
read kernel and system configuration information from /usr/lib/rmb/rmbbootinfo
read /usr/lib/rmb/rmbrc
read local $HOME/.rmbrc
allocate and initialize BASIC/UX workspace
scan /dev/rmb for device files
Is −b on command line?
yes
no
print boot screen
run any AUTOST program specified or found in $HOME
BASIC/UX is ready to use

HP BASIC/UX Boot Process

You can also indicate the autostart file in the `rmb` command (see the *HP-UX Reference*).

The AUTOST program can PRINT statements, LOAD files, and RUN programs. The sample AUTOST file below greets the user and then loads and runs a program.

```
10   PRINT "Welcome to the System."
20   PRINT
30   INPUT "Press RETURN to start the MONITOR program",C$
40   LOAD "MONITOR",1        !load and automatically run MONITOR
50   END
```

When you press Return as prompted, the file **MONITOR** is loaded and run.

To edit your own AUTOST program, enter the EDIT mode and create a program that fits your needs. Then SAVE the file in your home directory, named AUTOST.

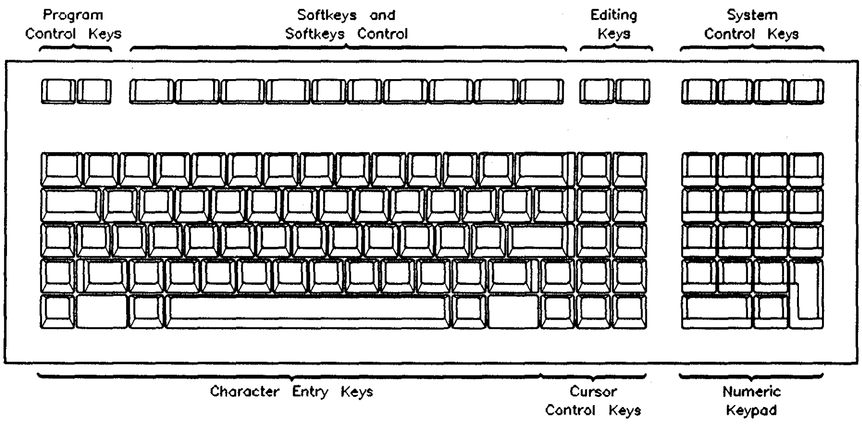
To avoid unnecessary errors, be sure to debug your AUTOST program before you re-enter BASIC/UX. The *HP-UX Reference* provides more information on the **rmb** command options, including how to specify your own AUTOST file.

11 Keyboard Information

This chapter provides a handy reference guide to BASIC/UX key definitions for the ITF keyboard. Note that other system programs may define the keys differently. Each key will be demonstrated where possible. The **cursor** referred to in the following paragraphs is the blinking-underline that points to a location on the screen. On some displays the cursor does not blink.

ITF Keyboards

The keys on the ITF keyboard are arranged into the following functional groups:



ITF Keyboard

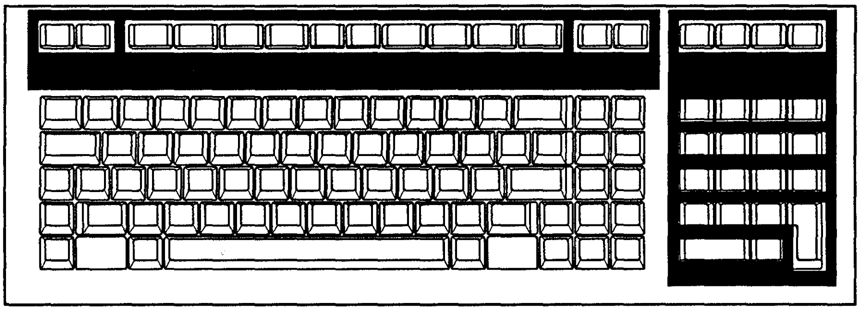
Note Before you proceed, type:

```
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:



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 the same process. 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:

- Enters data you provide when prompted by a program.
- Stores each line of code when typing in program lines.
- Executes commands entered on the keyboard input line.

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 the line. Type:

```
!FIRST LINE
```

and press Return. The computer accepts the statement as a program line and displays 20 in preparation for the next one. Press Stop (Shift-Stop) when finished.

Enter

Pressing Enter is the same as pressing the Return key.

Print (Shift-Enter)

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 you press Extend char plus another key, it enhances the character entry keys on Standard and European keyboards to print another character from the full 256-bit character set (see following table). On a Katakana keyboard, the "Roman" and "Katakana" keys select the other character sets.

Extended Character Set

Standard Letters[1]	Extended Characters		Standard Numbers & Symbols	Extended Characters	
	Lower Case	Upper Case		Lower Case	Upper Case
A	å	Å	1	¡	¡
B	■	■	2	@	@
C	ç	Ç	3	#	#
D	đ	Đ	4	$\frac{1}{4}$	$^{\mathrm{I}}\mathrm{O}$
E	æ	Æ	5	$\frac{1}{2}$	$\frac{1}{2}$
F	ƒ	ƒ	6	^	^
G	¤	¤	7	\	\
H	¥	¥	8	[	{
I	˜	˜	9	]	}
J	$	$	0	¿	¿
K	¢	¢	‘	≪	≫
L	ℒ	ℒ	-	—	¯
M	º	º	=	±	±
N	ª	ª	[	°	°
O	ø	Ø	]	|	|
P	þ	Þ	\	μ	μ
Q	·	·	;	£	£
R	´	´	’	‘	’
S	ß	ß	,	<	<
T	`	`	.	>	>
U	¨	¨	/	—	—
V	§	§			
W	~	~			
X	š	Š			
Y	^	^			
Z	¶	¶			

[1]The accents produced by pressing $\boxed{\text{Extend char}}$ with $\boxed{\text{I}}$, $\boxed{\text{R}}$, $\boxed{\text{T}}$, $\boxed{\text{U}}$, or $\boxed{\text{Y}}$ appear on the CRT, but the cursor does not advance until you press the key for the letter over which the accent must appear. For example, to produce Ä (in Caps mode), press and hold $\boxed{\text{Extend char}}$, then press $\boxed{\text{U}}$. The ¨ appears with the cursor beneath it. Now press $\boxed{\text{A}}$ to complete the Ä character.

Tab	Press the Tab key to move the cursor forward to preset tabs. Press Shift-Tab to move 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 *HP BASIC 6.2 Language Reference* in the "Useful Tables" section.
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, press the Prev key to scroll the display down one page and press the Next key to scroll up one page. In edit mode, these keys move the scroll 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 (if you entered the first line in a previous exercise, just press Return to accept it):

```
10 !FIRST LINE
20 !SECOND LINE
30 !THIRD LINE
40 !FOURTH LINE
```

Try 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 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 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

Press Insert line to insert a new line above the cursor's current position (edit mode only).

Type EDIT, then press Return. Type 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

Press Delete line to delete 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

Press Insert char to set insert mode; characters you insert appear 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

Press Delete char to delete 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

Press unshifted-Clear line (labeled Clr→End on the overlay) to clear from the current cursor position to the end of the line.

Press Shift-Clear line (labeled Clr Ln on the overlay) to clear the keyboard line and message/results line.

Type 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

Press either the shifted or unshifted version of Clear display to clear 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.

Pause (Stop)	Press unshifted-Stop (labeled Pause on the overlay) to *pause* program execution after the current line. Press `Continue` (unshifted f2) in the System menu to resume program execution from the point where it was paused.
Stop (Shift-Stop)	Press Shift-Stop (labeled Stop on the overlay) to *stop* program execution after the current line. To restart the program, press `RUN` (unshifted f3) in the System menu.
Break	Press Break (labeled Clr I/O on the overlay) to pause program execution when the computer is performing or trying to perform an I/O operation. Press Break 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 Break cancels the I/O operation and pauses the program at the current line.
Reset	Press Reset to pause 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	Press unshifted-Key 1 (Recall) to recall 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.

Press f8 in the System menu to perform the same recall function as Key 1.

Key 2—Alpha/ Dump Alpha	Press unshifted-Key 2 (Alpha) once to turn on the alphanumeric display. Press it the second time to turn off the graphics display. (This key will have no effect unless you are in "separate alpha/graphics" mode. Refer to the SEPARATE ALPHA FROM GRAPHICS keyword in the *HP BASIC Language Reference*.)

Press Shift-Key 2 (Dump Alpha) to print a complete copy of the alpha display on the default printer. The Dump Alpha function is also executed by Print (Shift-Enter).

(Key 3)—Graphics/ Dump Graph

Press unshifted-(Key 3) (Graphics) once to turn on the graphics display. Press it the second time to turn off the alphanumeric display. (This key will have no effect unless you are in "separate alpha/graphics" mode. Refer to the SEPARATE ALPHA FROM GRAPHICS keyword in the *HP BASIC Language Reference*.)

Pressing (Shift)-(Key 3) (Dump Graph) prints a complete copy of the graphics display on the default printer. The combined alpha and graphics display will be printed unless you are in "separate alpha/graphics" mode.

(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.

Softkeys and Softkey Control

There are eight softkeys (labeled (f1) through (f8)) and two keys that control the definitions of the softkeys ((Menu) and (System)). Refer to chapter 4 for a discussion of menus and key labels.

When the BASIC system is booted, the softkeys default to System mode and the System menu appears at the bottom of your display. System softkeys are defined following control key definitions. In addition to the System mode, there are three User modes: User 1, User 2, and User 3. *HP BASIC Programming Guide* describes how to set up User modes.

Softkey Control Keys

(System)	To assume System mode, press unshifted-(System). . The System menu is displayed, *if* the (Menu) key is toggled to the *on* position.
(User)	Press (User) ((Shift)-(System)) to put the softkeys in User mode. A User menu is displayed *if* the (Menu) key is toggled to the *on* position. The system remembers which User menu you are 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 always goes to the User 1 menu. There are additional iterations with EDIT mode; see "Softkey Menu Changes" in the "Editing and Storing Programs" chapter for details.
(Menu)	Press unshifted-(Menu) to toggle the softkey labels; it turns them on if they're off and turns them off if they're on. Press (Shift)-(Menu) to increment User mode and menu *if* User mode is *on*.

Try the following exercises to learn how the two control keys work.

First, get the System mode selected and menu displayed. If the System menu 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 appears 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.

Press unshifted-System to return to the System menu.

System Softkeys

The following paragraphs define the eight System softkeys.

`Step`	`Step` (unshifted-f1) lets you execute one program line at a time. This is useful for debugging 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. *HP BASIC Programming Guide* explains how to select other printall devices.

Press `Print All` to turn on printall mode. Now type the following command:

```
PRINT "THIS IS A KEYBOARD OPERATION"  Return
```

Both the PRINT command and the message are displayed on the screen, (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. 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 *HP BASIC Programming Guide*).

`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" section of the *HP BASIC 6.2 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 f1 (described earlier). Recall recalls the last line that you entered, executed, or deleted. Several previous lines can be recalled this way. Recall is particularly useful 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.

Press Shift-f8 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.

Terminal Keyboard Reference

If you do not have an ITF keyboard, some keys may operate in a different way than those referenced in this document. In general, only the alphanumeric keys are guaranteed to work properly. The following describes:

- which terminal types are supported
- what keys on the terminal can be used to perform the same functions as those on the ITF keyboard
- what device can be used to input graphics from terminals.

Supported Terminal Types

The table below lists the terminal types supported on BASIC/UX. The HP Part Number is generally the same as the terminal type. The following terminal types without the **hp** prefix are also supported.

Terminal Types Supported

hp2392	hp2623
hp2393	hp2625
hp2394	hp2627
hp2397	hp2628
hp2622	hp150
hp262x	70092
70094	

Mapping Terminal Keys to ITF Keyboard Keys

To perform the same operation as some of the ITF keyboard keys on a terminal, or some of the BASIC Workstation keys (such as (Recall)),use the following mappings.

Terminal Key Mappings

BASIC System Key	Key Mapping
(Reset)	(CTRL)-(r)
(Clear I/O)	(CTRL)-(c)
(Clear line)	(CTRL)-(e)
(Recall)	(CTRL)-(l)
(Pause)	(CTRL)-(p)

These keyboard mappings can be turned on and off from the `rmb` command line. To turn the keyboard mappings on, type:

```
rmb -t
```

To turn the keyboard mappings off, do not use the `-t` option.

You can also turn keyboard mappings on and off by placing the environment variable `TERM_CONTROL` in your `$HOME/.rmbrc` file. To turn keyboard mappings on using this variable, type the following in your `$HOME/.rmbrc` file:

`TERM_CONTROL=on` *turns keyboard mapping on*

To turn keyboard mappings off using this variable, type the following in your `$HOME/.rmbrc` file:

`TERM_CONTROL=off` *turns keyboard mapping off*

Also, you must note the following:

- Break, Reset, aids, user keys, and modes keys are not recognized by BASIC/UX (they are local to the terminal).
- Keypad keys on terminals are not recognized by BASIC/UX.
- Shifted non-alphanumeric keys are not recognized by BASIC/UX.

Some Hints While Using Terminals

- There is no Stop key. Therefore, to stop a program, type

  STOP Return

  To exit the EDIT mode, use the Clear Display or Clear Line keys.
- You cannot cycle through the system softkey menus with softkey control keys. Use the SYSTEM KEYS, USER *n* KEYS, or CONTROL KBD,2 statements (see the *HP BASIC 6.2 Language Reference*).
- There is no RECALL key. Use Ctrl-G or cycle through the system softkeys for the "RECALL" softkey.

Graphics Input from Terminals

The mouse is not supported as a graphics input device for a terminal. From a terminal, you will have to use the keyboard arrow keys for graphics input — they are the only graphics input device available.

Index

HEWLETT
PACKARD

HP Part Number
E2040-90001

Printed in U.S.A. E0891

E2040-90601 Manufacturing Number