# BASIC/UX Programming Techniques

## Volume 2: Porting Information

**HP 9000 Series 300 Computers**
**BASIC/UX 5.5**

HP Part Number 98796-90010

HEWLETT PACKARD

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

# Printing History

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

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

November 1988...Edition 1

# Table of Contents

# Porting to 3.0 14

# Porting to 3.0 14

This chapter describes the differences between BASIC 2.0/2.1 extensions and BASIC 3.0.

**Note**

If you are porting a program from a "pre-3.0" version of the BASIC system to a 4.0 or 5.0 system, then you should also read the subsequent porting chapters. Anytime you see 3.0 mentioned in this chapter it also refers to all subsequent system versions.

## Porting Topics Covered

The following areas require consideration when transporting programs from BASIC 2.0/2.1 to BASIC 3.0. They are listed in the order in which they're discussed in this chapter.

- Compatibility with previous versions
- Configuring BASIC
- Statement changes
- CSUBs
- PHYREC
- Knob
- Graphics
  - Default plotter
  - Implicit GCLEAR
  - Input device viewport
  - Graphics Tablet DIGITIZE
  - The VIEWPORT Statement
  - The PIVOT Statement
- Display functions
- Prerun on LOADSUB
- Special case of I/O transfers

## Compatibility with Preceding Versions

If you have programs which were written on previous Series 200 BASIC systems, you can use these same programs with little or no changes. The major task you have to perform is to configure the BASIC 3.0 system with the necessary BIN files.

## Configuring BASIC

This section contains procedures that help you ensure you have loaded all the required language extensions and drivers. It also tells you where to find related information in your BASIC manual set.

### Helpful Documentation

The BASIC manuals can help you determine which BIN files you need. The chapter entitled "Language, Extensions, Drivers and Configuration" of *Installing and Maintaining Your BASIC System* contains a brief description of each BIN file. It also lists the functions and statements supported by each Language Extensions BIN file.

The "Language History" section of the *BASIC Language Reference* manual contains an alphabetical list of all keywords showing which BIN file, if any, is needed for each keyword. The Keyword Dictionary in the *BASIC Language Reference* manual also indicates which BIN file is required for each keyword. Keep in mind that some keywords are partially supported by just core BASIC (SYSTEM_BA*n*) and that additional capabilities may require a BIN file. The Keyword Dictionary uses shading in the syntax diagram to show which aspects of a statement require an additional BIN file. For example, CAT is supported by core BASIC, but the MS BIN file is needed to support SELECT and other advanced features.

## Missing Language Extensions BIN Files

Follow this procedure to make sure that you have all the language extensions BIN files that a program needs. The procedure ensures that each program unit is not prerun and then preruns all program units. Prerun reports the first missing BIN file that it finds. Editing a program unit ensures that it is not in the prerun state. Stepping a stopped program preruns it.

Load the program and the BIN files PDEV and ERR. Enter the first line of the program to ensure that the main program is not in a prerun state. Find every SUB statement (using the FIND command enabled by the PDEV BIN file) and enter it. Find every DEF FN statement and enter it. Now no program unit is in a prerun state. Stepping preruns every subprogram. If prerun finds a statement or option that requires a missing BIN file, error 1 is given along with the name (if the ERR BIN file is loaded) of the missing BIN file. After loading the missing BIN file, step again to prerun the program. If a BIN file is missing, error 1 and its name are given. Repeat this process until stepping gives no errors. At that point, all language extensions BIN files needed by the program are present. If the program loads subprograms or other programs, repeat this process for each of them.

This process does not work for a secured program. The best approach in this case is to ask the author or vendor for a list of the BIN files required. If this is not possible, load the ERR BIN file and run the program. Whenever a statement is executed that requires a missing BIN file, an error 1 and the name of the BIN file are given. After loading the BIN file, the program can be continued. However, it may be difficult to force the execution of all paths in the program. This can be a serious problem if a real-time control program is surprised by a missing BIN file at a critical moment.

Remember, if you have enough memory, you can load all the BIN files. **However, only load KNB2_0 if you want KNOBX to function as it does in BASIC 2.0/2.1 and KNOBY to always return a zero. Refer to the Knob section later in this chapter for more information.**

## Missing Driver BIN Files

To ensure that all required driver BIN files are loaded, load the appropriate BIN file for each interface card and I/O port used (including the built-in HP-IB and RS-232 serial interface, if present). Also load the appropriate disc driver BIN file for each disc drive used.

If an operation is attempted to a device but the card driver BIN file is missing, the message "ERROR 163  I/O interface/driver not present" is usually provided. Examples of this are: CAT":,700" or PRINTER IS 701 with the HPIB BIN file missing.

If the card BIN file is present but the disc driver BIN file is missing, an attempt to access the disc causes error 1. If the ERR BIN file is loaded, the message "ERROR 1 Configuration error" is provided.

If both the card driver and disc driver BIN files are missing, error 163 is usually given but error 1 can also occur.

## Statement Changes

There are several statements added with BASIC 3.0. These are listed below.

```
KNOBY              PRINTER IS file
LIST BIN           READ LABEL
MAXREAL            RES
MINREAL            SCRATCH BIN
MODULO             SECURE
PDIR               SET LOCATOR
PLOTTER IS file    STORE SYSTEM
PRINT LABEL        SYSBOOT
```

Two statements were deleted, STORE BIN and RE-STORE BIN.

## CSUBs

If you used Pascal-compiled subprograms (CSUBs) in your BASIC 2.0/2.1 programs, you need to purchase a Pascal 3.0 system upgrade and a CSUB Utility upgrade to use those CSUBs with BASIC 3.0. You must recompile the Pascal routine on Pascal 3.0 and re-execute the CSUB utility to make the routine look like a BASIC subprogram. If you are using a CSUB supplied by a vendor, you must have the supplier update the CSUB for you.

## PHYREC

The PHYREC routine that allowed you to read from and write to physical records on a disc was changed from a binary program to a CSUB with BASIC 3.0. The PHYREC CSUB is located on the *BASIC Utilities Disc 1.*

You must append the PHYREC CSUB to your program and change the PHYREAD/PHYWRITE statements. If the PHYREC binary is appended to a program, a warning message is displayed and the binary is ignored by BASIC.

Use the following steps to locate all the lines for an application that uses PHYREC and change them to call and append the PHYREC CSUB.

1. Boot a BASIC 2.0/2.1 system.
2. Delete the PHYREC binary.

   `LOAD "program"`
   `SAVE "program2"` - This saves the program without the binary.
   `SCRATCH A` - This deletes the program and binary from memory.
   `GET "program2"` - Calls to PHYREC are commented. Write down the line numbers.
   `RE-STORE "program"`
   `PURGE "program2"`

3. Attach the PHYREC CSUB.

   ```
   LOADSUB ALL FROM "PHYREC"
   ```

   This file is located on *BASIC Utilities Disc 1.* **Do not try to run your application until you have completed all steps.**

4. Uncomment and change all the calls to PHYREC. These are the lines you noted in step 2 above.

   ```
   PHYREAD Sector,Int_array(*) > Phyread(Sector,Int_array(*))
   PHYWRITE Sector,Int_array(*) > Phywrite(Sector,Int_array(*))
   ```

5. If `Sector` is declared to be an INTEGER, you need to put it into parentheses so that PHYREC will interpret is as a REAL.

   ```
   Phyread((Sector),Int_array(*))
   ```

6. The syntax for a conditional call must be changed from:

   ```
   IF condition THEN PHYREAD Sector,Int_array(*)
   ```

   to:

   ```
   IF condition THEN
     Phyread(Sector,Int_array(*))
   END IF
   ```

   or to:

   ```
   IF condition THEN CALL Phyread(Sector,Int_array(*))
   ```

7. `RE-STORE "program"` after you have completed the changes.
8. Boot BASIC 3.0 and run your application.

## Knob

In BASIC 3.0, unshifted knob movement causes horizontal cursor movement, and shifted knob movement results in vertical movement. This allows for greater compatibility between the knob and the HP-HIL mouse. (In BASIC 2.0/2.1, horizontal and vertical modes are toggled and interlocked.)

### The KNOBX Function

The BASIC 2.0/2.1 definition of KNOBX, which we will refer to as all-pulse mode, is as follows: When an ON KNOB statement is executed to trap knob movement, knob pulses are accumulated and accessed via the KNOBX statement. Since the KNOBX function returns information on X-axis movement, a method of tracking Y-axis movement is not directly available with BASIC 2.0/2.1. The common method used to track Y-axis movement, is to interrogate keyboard status register 10 for information on the state of the CTRL and SHIFT keys at the time of the last knob interrupt. Using this information, SHIFTed and/or CTRLed knob movement could be interpreted differently; in fact, an example program showing this was included in the 2.0/2.1 manual set. Following is another sample 2.0/2.1 program with this type of knob interpretation:

```
          :
          :
 30    ON KNOB .1 GOSUB Knobsvc
 40    Loop:  GOTO Loop
 50    STOP
 60 !
 70    Knobsvc:  !
 80       STATUS KBD,10;State          ! was SHIFT or CTRL key pressed?
 90       Shift=BIT(State,0)           ! bit 0 set = SHIFT key pressed
100       Ctrl=BIT(State,1)            ! bit 1 set = CTRL key pressed
110       SELECT Shift
120          CASE 0                ! if shift not pressed, X direction
130             IF Ctrl THEN       ! if ctrl pressed, give finer resolution
140                X=X+KNOBX/10
150             ELSE
160                X=X+KNOBX
170             ENDIF
180          CASE 1                ! if shift pressed, Y direction
190             IF Ctrl THEN       ! if ctrl pressed, give finer resolution
200                Y=Y+KNOBX/10
210             ELSE
220                Y=Y+KNOBX
230             ENDIF
240       END SELECT
          :
          :
```

With the introduction of the new HP-HIL keyboards (no built-in knob but optional mouse), the intent was to allow the mouse to emulate knob behavior in situations where a knob is no longer present. The all-pulse mode of interpretation, however, is unacceptable when using a mouse because the mouse is not a unidirectional device, yet movement information in only one direction is available. It is virtually impossible to move the mouse in one direction only. To be able to distinguish movement in each direction, the keyword KNOBY has been added to BASIC 3.0. KNOBY returns the net number of Y-direction knob pulses counted since the last time the KNOBY counter was zeroed.

## Keyboards with Built-in Knob

To convert your programs which run on hardware with a built-in knob from 2.0/2.1 to 3.0, simply replace KNOBX with KNOBX+KNOBY in situations where total knob movement is being recorded. The major difference in 3.0 operation is that knob pulses in the X-direction are accessed via KNOBX and knob pulses in the Y-direction are accessed via KNOBY. One way to modify the above program for 3.0 is:

```
           :
           :
  30    ON KNOB .1 GOSUB Knobsvc
  40    Loop:  GOTO Loop
  50    STOP
  60 !
  70    Knobsvc:  !
  80       STATUS KBD,10;State           ! was SHIFT or CTRL key pressed?
  90       Shift=BIT(State,0)            ! bit 0 set = SHIFT key pressed
 100       Ctrl=BIT(State,1)             ! bit 1 set = CTRL key pressed
 110       SELECT Shift
 120          CASE 0                 ! if shift not pressed, X direction
 130             IF Ctrl THEN        ! if ctrl pressed, give finer resolution
 140                X=X+KNOBX/10
 150             ELSE
 160                X=X+KNOBX
 170             ENDIF
 180          CASE 1                 ! if shift pressed, Y direction
 190             IF Ctrl THEN        ! if ctrl pressed, give finer resolution
 200                Y=Y+KNOBY/10
 210             ELSE
 220                Y=Y+KNOBY
 230             ENDIF
 240       END SELECT
           :
           :
```

However, this does not work with the HP-HIL mouse. A method that works with the HP-HIL mouse as well as with the built-in knob is:

```
              :
              :
   30     ON KNOB .1 GOSUB Knobsvc
   40     Loop:  GOTO Loop
   50     STOP
   60 !
   70     Knobsvc:  !
  160        X=X+KNOBX
  170        Y=Y+KNOBY
              :
              :
```

## HP-HIL Keyboards with Mouse

If your ON KNOB routine reads keyboard status register 10 for shift-knob or control-knob actions you will need to make some other changes to convert 2.0/2.1 programs to 3.0. On HP-HIL input devices (i.e. the mouse), keyboard status register 10 has a different interpretation: bit 0 (SHIFT key pressed) is set if last data processed at the last knob interrupt was Y-axis information (data accessed via KNOBY) and cleared if last data processed was X-axis data; bit 1 (CTRL key pressed) is never set. If unidirectional HP-HIL devices were to become available, a toggle switch would exist on the device to switch between X-axis and Y-axis directions and the shift bit on keyboard status register 10 would be set when in the Y-direction mode.

The previous program segment shows recommended servicing of the mouse.

## Programming for Both Versions and Keyboards

In the most complicated case, you may wish to write code that runs on both BASIC 2.0/2.1 and BASIC 3.0 with either a built-in knob or HP-HIL mouse. Write knob service routines for the BASIC 2.0/2.1 program and the BASIC 3.0 program and LOADSUB the appropriate routine based on the current version of BASIC. The following program segments show one method of handling this situation:

```
          :
          :
30   GOSUB Whichversion
40   IF Version=3 THEN
50      LOADSUB ALL FROM "KNOBSVC3_0"
60   ELSE
70      LOADSUB ALL FROM "KNOBSVC2_0"
80   END IF
          :
          :
110  Whichversion:        ! running BASIC 2.0/2.1 or 3.0 ?
120     ON ERROR GOTO B2_0
130     STATUS 2,2;A      ! KBD register 2 does not exist for 2.0/2.1, error
140     Version=3         ! if line 130 didn't error out, must be 3.0
150     GOTO Versionfound
160  B2_0: !
170     Version=2
180  Versionfound: !
190     OFF ERROR
200     RETURN
          :
          :
```

## KNB2_0

Because these modifications to the KNOB facilities may prevent your 2.0/2.1 programs from running on BASIC 3.0 without making a few changes, we have developed a way to return to the all-pulse mode of KNOB operation in which all knob pulses are accessed via KNOBX. **This mode is not recommended for the HP-HIL mouse.** To switch to this mode, execute CONTROL KBD,11;1.

---

**Note**

If you select all-pulse mode, KNOBY always returns a zero.

---

Executing CONTROL KBD,11;0 returns you to the 3.0 mode of operation in which Y-direction pulses are accessed via KNOBY. To determine the mode, execute STATUS KBD,11;M. If M=0, KNOBX is in horizontal-pulse mode; if M=1, KNOBX is in all-pulse mode.

In some cases, it may be desirable to make this mode change implicitly. This can be accomplished by loading the BIN file KNB2_0 from the *Language Extensions* disc. A LIST BIN describes the new BIN file as `2.0 KNOBX Definition`. The only effect of KNB2_0 being loaded is that it executes CONTROL KBD,11;1 for you automatically. When KNB2_0 is loaded, executing SCRATCH A also automatically executes CONTROL KBD,11;1. Note that if this binary is included in a stored system (e.g. created with the STORE SYSTEM statement), the effects are the same as loading it afterwards.

---

**Note**

All-pulse mode (KNB2_0 loaded) is not recommended for the HP-HIL mouse. Also note that the KNB2_0 binary and the all-pulse mode are not supported by BASIC/UX.

---

### Special Consideration

Since the KNB2_0 binary functionality is not supported by BASIC/UX, you will need to modify any program containing this functionality.

## Graphics

Several graphics statements function differently with BASIC 3.0 than they did in BASIC 2.0/2.1. This section explains the differences.

### Default Plotter

The initialization of graphics system variables and devices was changed slightly in BASIC 3.0. When GINIT is executed, several operations are performed automatically such as setting line type and character size. In addition to these operations, BASIC 2.0/2.1 also implicitly does a PLOTTER IS 3,"INTERNAL" to select the CRT as the default plotting device. In BASIC 3.0, the default plotting device is not selected until a statement is executed that affects it (e.g., DRAW, LABEL, GLOAD). At this time, the appropriate PLOTTER IS statement is executed along with GCLEAR, VIEWPORT and WINDOW statements. Refer to GINIT in the *BASIC Language Reference* manual for more information.

### Implicit GCLEAR

In BASIC 2.0/2.1, any graphics statement following GINIT except PLOTTER IS, GINIT, and DUMP DEVICE causes the implicit execution of GCLEAR, VIEWPORT, and WINDOW. With BASIC 3.0, if a statement that requires a plotter is executed after GINIT, a PLOTTER IS CRT, "INTERNAL" is executed followed by GCLEAR, VIEWPORT, and WINDOW. Refer to GINIT in the *BASIC Language Reference* manual for more information.

### Input Device Viewport

The GRAPHICS INPUT IS statement sets the hard clip limits of the input device to the largest space possible that has the same aspect ratio as the output device. Since this was not so in earlier versions, there were two potential problems. The first problem is that it is possible to move to positions on the input device that do not exist on the output device. The extent of this problem may be reduced with BASIC 3.0, but the problem is not eliminated. The second problem is that the aspect ratios of the input and output devices may differ causing pictures on the devices to appear different. BASIC 3.0 solves this problem by automatically setting the hard clip limits of the input device to the largest possible space that has the same aspect ratio as the output device.

## Graphics Tablet DIGITIZE

A stylus press on the HP 9111A Graphics Tablet prior to execution of a DIGITIZE statement does not satisfy the DIGITIZE with BASIC 3.0 as it does with BASIC 2.0/2.1. An output of the string "SG" to the graphics tablet after the GRAPHICS INPUT IS statement causes BASIC 3.0 to work like BASIC 2.0/2.1.

## The VIEWPORT Statement

VIEWPORT was changed in BASIC 3.0 to make it compatible with the Series 500 and the industry standard. In BASIC 3.0, VIEWPORT rescales immediately. In BASIC 2.0/2.1, VIEWPORT does not rescale; only WINDOW and SHOW statements rescale.

An example helps demonstrate the difference. The following program behaves the same way in BASIC 2.0/2.1 and 3.0 because it does not have a VIEWPORT statement. It draws a large frame with a large quadrangle in it as shown in the following figure titled "BASIC 2.0/2.1 and 3.0 without VIEWPORT".

```
10   GINIT
20   GRAPHICS ON
30   FRAME
40   CLIP OFF
50   MOVE 0,50
60   DRAW 100,100
70   DRAW RATIO*100,50
80   DRAW 100,0
90   DRAW 0,50
100  END
```

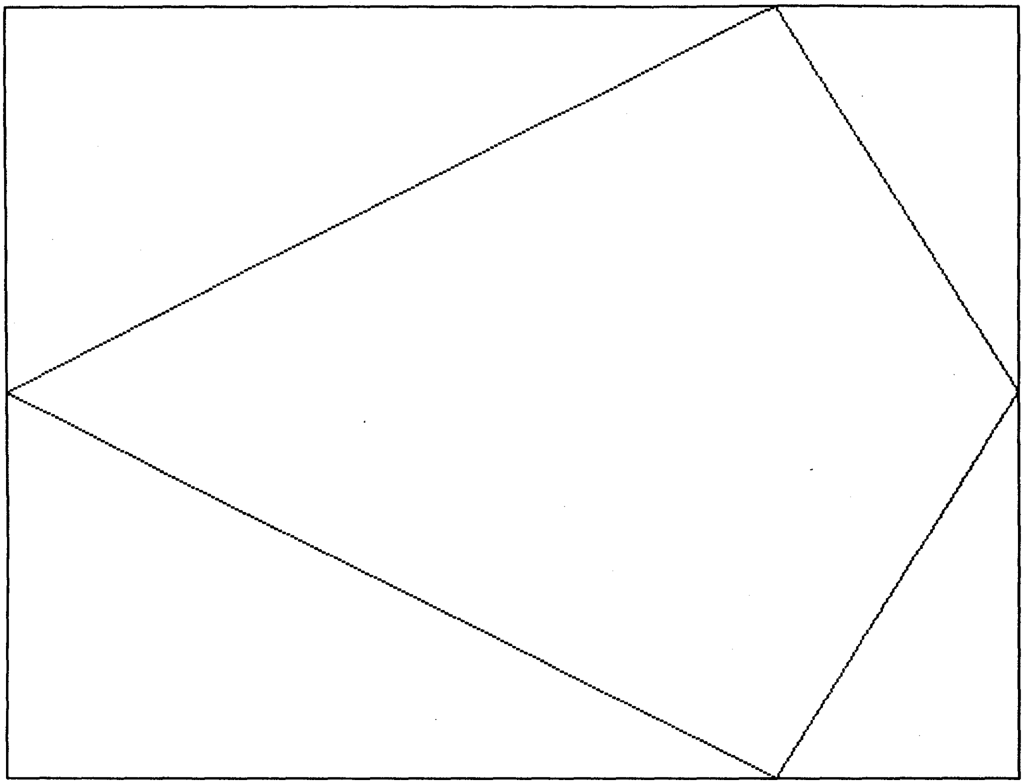


**Figure 14-1. BASIC 2.0/2.1 and 3.0 without VIEWPORT**

If a VIEWPORT statement is placed in the program, BASIC 2.0/2.1 and BASIC 3.0 give different results. The program becomes:

```
10   GINIT
20   GRAPHICS ON
30   VIEWPORT 80,100,20,80
40   FRAME
50   CLIP OFF
60   MOVE 0,50
70   DRAW 100,100
80   DRAW RATIO*100,50
90   DRAW 100,0
100  DRAW 0,50
110  END
```

With BASIC 2.0/2.1, the result is a small frame with a large quadrangle around it (see figure titled "BASIC 2.0/2.1 with VIEWPORT"). The frame is what one would expect from the VIEWPORT; it is tall and thin. The quadrangle is the same as the one drawn by the program without the VIEWPORT because the VIEWPORT has not caused the DRAW's to be rescaled.

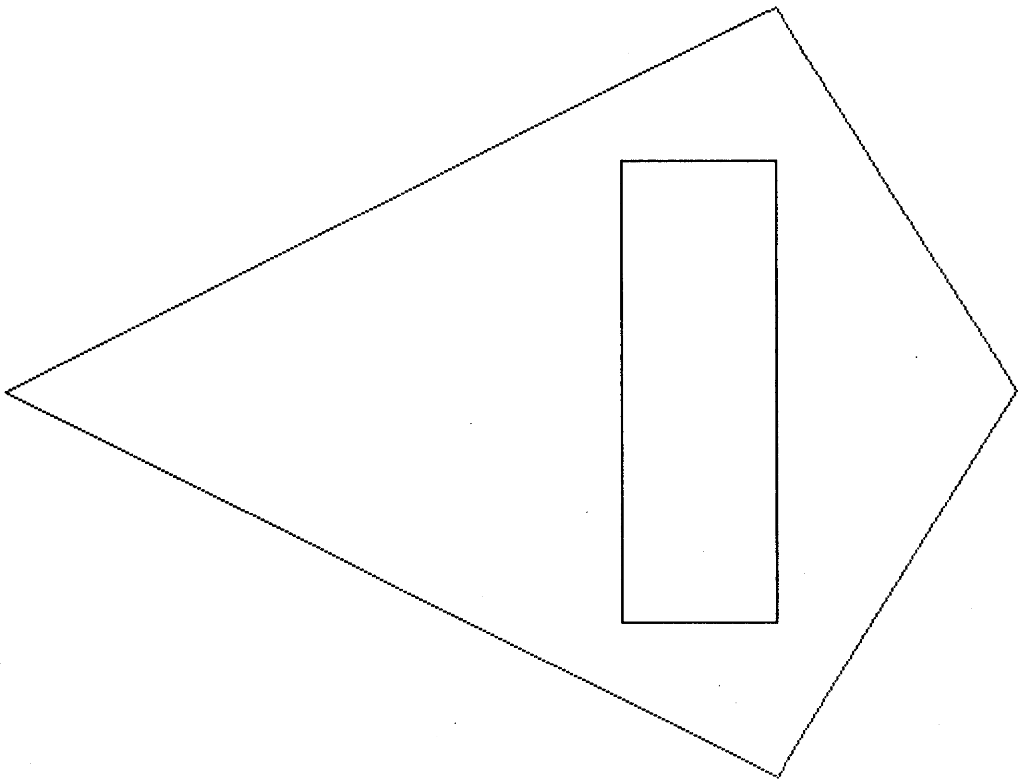


**Figure 14-2. BASIC 2.0/2.1 with VIEWPORT**

With BASIC 3.0, the result is a small frame with a small quadrangle inside the frame (see figure titled "BASIC 3.0 with VIEWPORT"). The frame is the same frame as given by BASIC 2.0/2.1. The quadrangle fits inside the frame because the VIEWPORT in BASIC 3.0 causes all subsequent DRAW's to be rescaled.

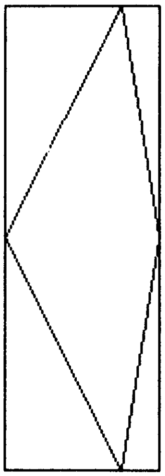


**Figure 14-3. BASIC 3.0 with VIEWPORT**

The VIEWPORT change usually does not affect programs because most programs used a sequence such as:

```
VIEWPORT 20,100,20,80
WINDOW Xmin,Xmax,Ymin,Ymax
```

The result of these two statements in order is the same in BASIC 2.0/2.1 and BASIC 3.0.

Some BASIC 2.0/2.1 programs used the following order:

```
VIEWPORT 20,100,20,80
WINDOW Xmin,Xmax,Ymin,Ymax
VIEWPORT 0,100*RATIO,0,100
```

The second VIEWPORT was used to change the soft clip limits. In BASIC 2.0/2.1, the second VIEWPORT did not rescale so that the scale defined by the WINDOW and the first VIEWPORT remains effective. When the above sequence is run in BASIC 3.0, the second VIEWPORT rescales all subsequent plotting.

The best solution to this problem is to change the sequence to:

```
VIEWPORT 20,100,20,80
WINDOW Xmin,Xmax,Ymin,Ymax
CLIP OFF
```

## The PIVOT Statement

In BASIC 3.0, the local origin of RPLOT and LABEL is affected by the PIVOT statement. The best way to see the differences between BASIC 2.0/2.1 and BASIC 3.0 is by studying the following examples.

### RPLOT with PIVOT

The following program illustrates the effects of PIVOT on RPLOT statements. Outputs of the program with BASIC 2.0/2.1 and 3.0 are shown after the program.

```
10      DEG
20      GINIT
30      GRAPHICS ON
40      VIEWPORT 0,64,51,100
50      Pivot(0)
60      VIEWPORT 66,130,51,100
70      Pivot(30)
80      VIEWPORT 0,64,0,49
90      Pivot(60)
100     VIEWPORT 66,130,0,49
110     Pivot(90)
120     END
130     SUB Pivot(P)
140     WINDOW 0,131,0,100
150     FRAME
160     MOVE 30,80
170     LABEL "PIVOT",P
180     MOVE 40,20
190     PIVOT P
200     Tri
210      MOVE 80,20
220     Tri
230     PIVOT 0
240     SUBEND
250     SUB Tri
260     RPLOT 20,0,-1
270     RPLOT 20,20
280     RPLOT 0,0
290     SUBEND
```

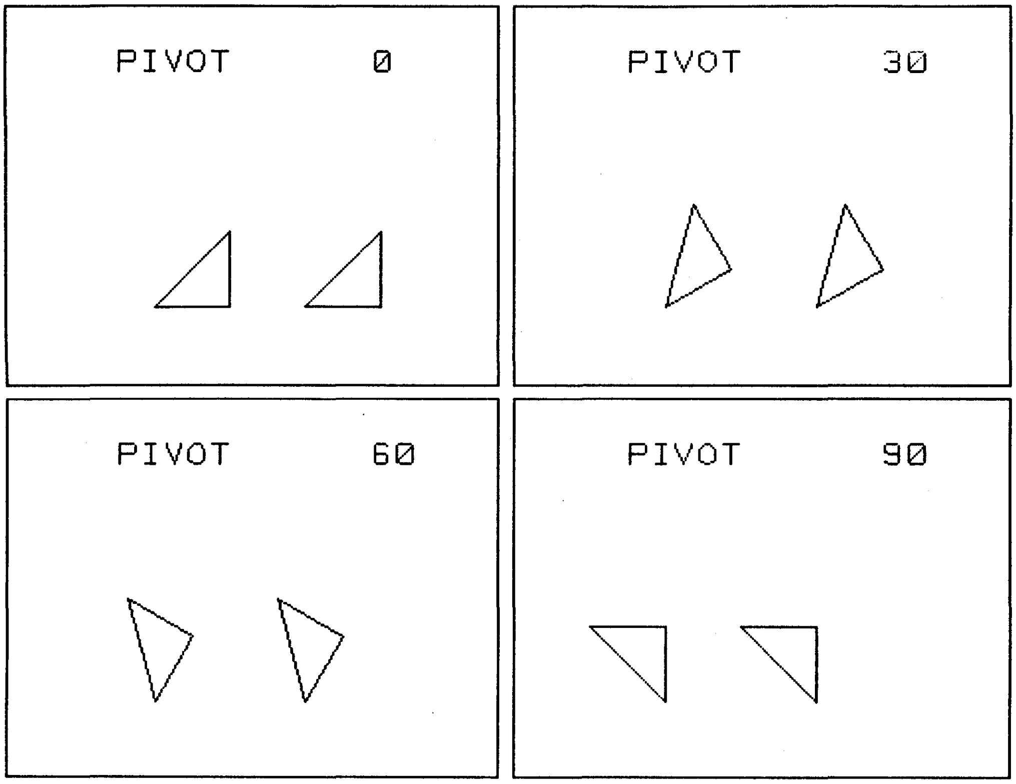


**Figure 14-4. BASIC 2.0/2.1 RPLOT with PIVOT**

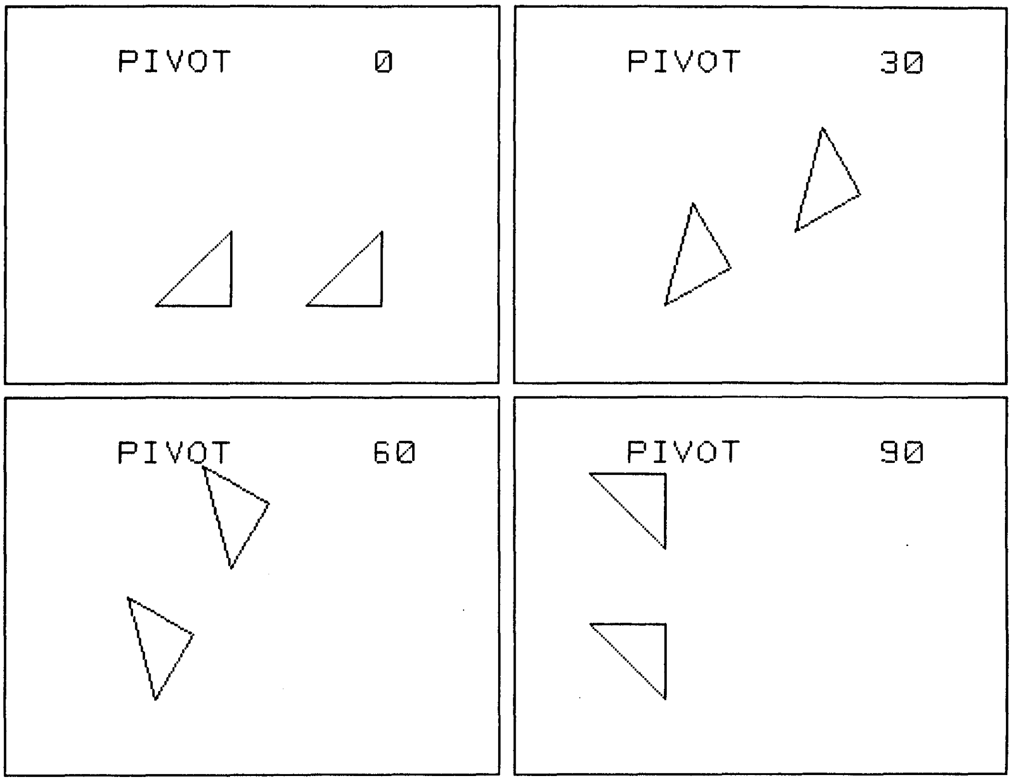


**Figure 14-5. BASIC 3.0 RPLOT with PIVOT**

## LABEL with PIVOT

The following program illustrates the effects of PIVOT on LABEL statements. Outputs of the program with BASIC 2.0/2.1 and 3.0 are shown after the program.

```
10      DEG
20      GINIT
30      GRAPHICS ON
40      VIEWPORT 0,64,51,100
50      FRAME
60      Pivot(0)
70      VIEWPORT 66,130,51,100
80      FRAME
90      Pivot(30)
100     VIEWPORT 0,64,0,49
110     FRAME
120     Pivot(60)
130     VIEWPORT 66,130,0,49
140     FRAME
150     Pivot(90)
160     END
170     SUB Pivot(P)
180     WINDOW 0,131,0,100
190     MOVE 40,80
200     LABEL "PIVOT",P
210     MOVE 60,60
220     PIVOT P
230     IDRAW 0,0
240     LABEL "L1"
250     LABEL "L2"
260     LABEL "L3"
270     IDRAW 0,0
280     PIVOT 0
290     IDRAW 0,0
300     LABEL "L4"
310     LABEL "L5"
320     LABEL "L6"
330     SUBEND
```

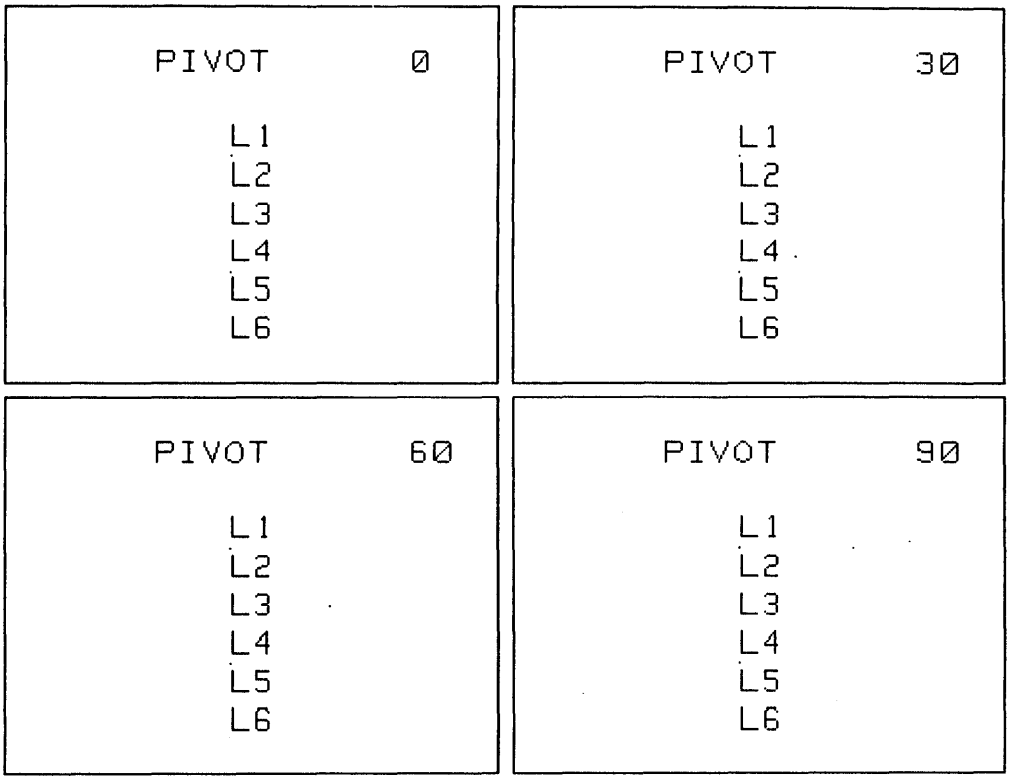


**Figure 14-6. BASIC 2.0/2.1 LABEL with PIVOT**

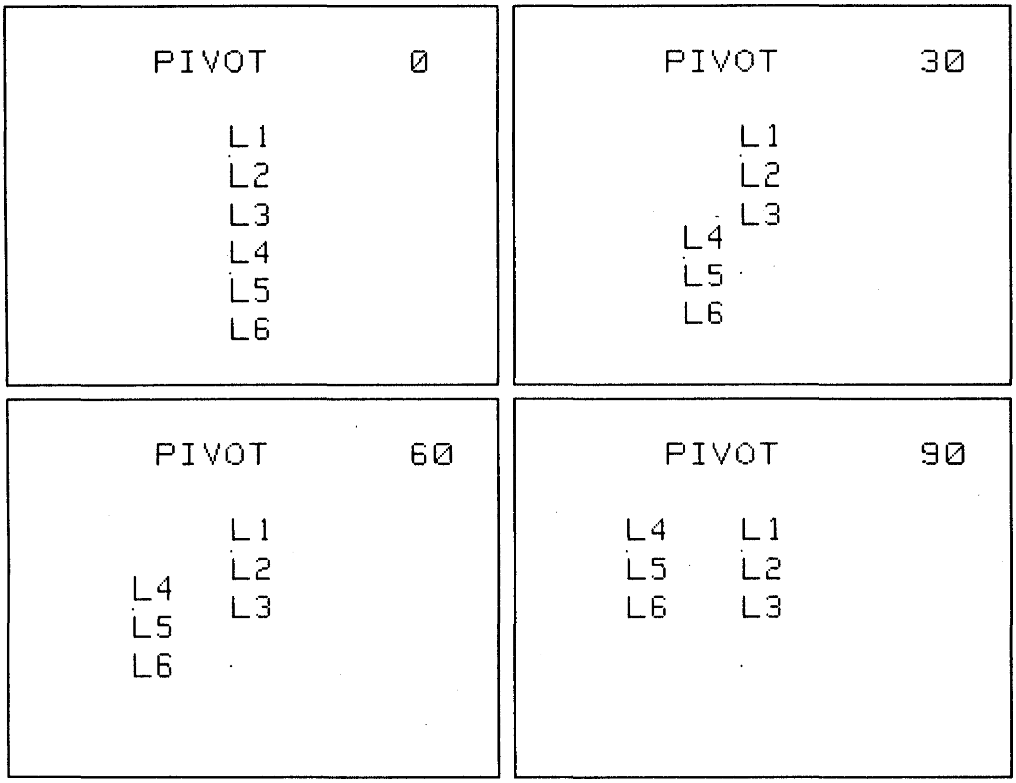


**Figure 14-7. BASIC 3.0 LABEL with PIVOT**

## Display Functions

The effect of turning Display Functions mode on is to display special control characters on the screen. In BASIC 2.0/2.1, Display Functions has no effect on control characters 128 through 159. With BASIC 3.0, the appropriate character is displayed on the screen when control characters 128 through 159 are displayed and Display Functions is enabled. For example, on a Model 236 running BASIC 2.0/2.1, the following statement:

```
PRINT CHR$(129)&''HI THERE"&CHR$(128)
```

results in:

`HI THERE` (in inverse video)

With BASIC 3.0, the result is:

$^{h}{}_{p}$ `HI THERE` $^{h}{}_{p}$ $^{C}{}_{R}$
$^{L}{}_{F}$

The $^{h}{}_{p}$ symbols are machine dependent; the actual characters displayed may vary with other models.

## Prerun On LOADSUB

To speed the execution of the LOADSUB statement, BASIC 3.0 does not prerun each subprogram loaded by the execution of the LOADSUB statement if the subprogram has been stored in a "prerun state". This differs from BASIC 2.0/2.1 in that BASIC 2.0/2.1 does prerun on the entire program every time LOADSUB is executed. The only effect seen by this change is improved performance when loading subprograms with the LOADSUB statement. For more information on prerun, refer to the "Program Structure and Flow" chapter.

## Special Case of I/O Transfers

A special case of decreased I/O performance has occurred with BASIC 3.0 due to a missed interleave caused by the increased overhead for handling multiple processors. Outbound transfers without DMA to the 913xA/B/V/XV Winchester disc drives perform at 11.75 Kbytes/second in BASIC 3.0. In BASIC 2.0/2.1, those transfers perform at a rate of 50 Kbytes/second. This degradation occurs only if all the following conditions are met:

- 8 MHz processor board (no cache)
- Not using DMA
- Using outbound TRANSFER (not OUTPUT) to 913xA/B/V/XV drive

This performance degradation affects users who are logging test data onto their discs. Adding DMA can increase the outbound transfer rate to 50 Kbytes/second. (Inbound transfers without DMA from those drives perform at 11.75 Kbytes/second in both BASIC 2.0/2.1 and BASIC 3.0.)

# Porting to Series 300 and 4.0 15

# Porting to Series 300 and 4.0 15

This chapter mainly focuses on one objective:

- Making BASIC programs which have been written for Series 200 computers run on Series 300 computers. (This process is known as "porting" programs.)

---

**Note**

If you are porting from a "pre-3.0" version of BASIC to the 4.0 version, then you should also read the preceding "Porting to 3.0" chapter. Anytime you see 4.0 mentioned in this chapter it also refers to all subsequent system versions.

---

This chapter also discusses the following topics, which may not in all cases be directly related to porting existing Series 200 software to Series 300 computers:

- Configuring the built-in 98644-like RS-232C serial interface in Series 300 computers.
- Using the "98203 keyboard compatibility" mode with ITF keyboards (such as the 46020 keyboard).
- Using the 98546 Display Compatibility Interface in your Series 300 computer (this interface provides the alpha and graphics capabilities of the Model 217 computer).

---

**Note**

If you are not porting BASIC 3.0 programs to Series 300 computers (that is, if you will be using BASIC 4.0 on a Series 200 computer), then much of the first part of this chapter may not pertain to you. The subsequent sections called "Modifying the Source Program (Porting to 4.0)" and "BASIC 4.0 Enhancements for Series 200 Computers" will contain useful information for you.

---

## Methods of Porting

Here are several methods of porting Series 200 software to Series 300 machines:

- Just load the program into a Series 300 computer—with no modifications—and run it.
- Write and run a program that properly configures the Series 300 computer for the program.
- Make your Series 300 computer emulate a Series 200 Model 217 computer (by installing a HP 98546 Display Compatibility Interface), and then run your unmodified Series 200 program on it.
- Modify your Series 200 BASIC source program, and then run it on a Series 300 computer with the BASIC 4.0 system.

Each method has a slightly different set of requirements for its use, as described subsequently.

## Chapter Organization

This chapter is organized according to the above strategies. It consists of the following sections:

- Description of Series 300 computer hardware, focusing on the enhancements to and differences from Series 200 computers
- Descriptions of porting methods, including when and how to use each[1]:
  - Just loading and running programs
  - Using configuration programs
  - Using the "Display Compatibility Interface"
  - Modifying the program's source code

[1] Note that you may need to use more than one method in porting a program. For instance, you may need to write a configuration program *and* use the Display Compatibility Interface in order to port a program.

## Description of Series 300 Hardware

Acquiring a general understanding of the enhancements or changes to Series 200 computers provided by Series 300 computers will help you to choose a porting method.

### Areas of Change

Series 300 computers have changes in the following areas:

- Many choices of processor, display, and human interface boards:
  - Six displays (including a separate, high-speed display controller)
  - Two processors: MC68010, and MC68020 (with MC68881 math co-processor)
  - Battery-backed, real-time clock
  - RS-232C serial interface (similar to the 98644 serial interface)
  - ITF keyboard ("Integrated Terminal Family" keyboard, which is similar to Models' 217 and 237 keyboards, but different from other Series 200 models' keyboards)
- No ID PROM (not all Series 200 Models had this feature)

### Areas that Did Not Change

It will probably be comforting to know that if a feature is not listed above (and discussed in this chapter), then it is the same for both Series 300 and Series 200 computers.

It may also be comforting to note that Series 300 computers can use most Series 200 accessories and peripheral devices. See the *HP 9000 Series 300 Configuration Reference Manual* for a complete list.

## Displays

Series 300 display technology is the most visible area of change from Series 200 computers.

All Series 300 computers utilize bit-mapped alpha display technology, which *combines* alpha and graphics like the display of the Series 200 Model 237. (All other Series 200 models have *separate* alpha and graphics.)

The main difference between "non-bit-mapped" and "bit-mapped" alpha displays lies in whether or not alpha and graphics are separate.

- With non-bit-mapped alpha displays, alpha is *separate* from graphics. Alpha is produced by character-generating hardware, while graphics are produced by bit-mapping hardware.

  (You can use the ALPHA and GRAPHICS keys to turn on alpha and graphics independently. When alpha is already on, pressing the ALPHA key turns off graphics. Similarly, pressing the GRAPHICS key while graphics is on turns off alpha.)

- With bit-mapped alpha displays, alpha and graphics are *not* separate. Both alpha and graphics are produced by a combination of software and bit-mapping hardware.

  (With BASIC 4.0, there is a way to configure the Series 300 *color* displays as separate alpha and graphics planes. This technique is described in the subsequent "Using a Configuration Program" section.)

An effect of bit-mapped alpha is that both alpha and graphics are dominant. In other words, displaying a character on the screen overwrites all pixels within the character cell; the previous contents of those pixels, which may have been graphics, are lost. Also, any scrolling/clearing of the alpha screen will scroll/clear the graphics information on the screen, since they share the same display plane. Conversely, graphics operations overwrite alpha-related pixels.

With Series 300 computers, you may choose from one of six displays: monochrome and color, each available in both medium- and high-resolution versions[1]. (Most Series 200 computers have only one display available for each model.)

- Medium-resolution graphics displays have 512[2] horizontal by 400[3] vertical pixels (many of the Series 200 graphics displays had 512×390-pixel graphics displays).

  Alpha capabilities of these medium-resolution displays are 80 columns of characters by 26 lines on-screen, plus 51 lines off-screen (as opposed to the 80×25-character alpha displays, with 39 lines off-screen, of many Series 200 computers). The characters on Series 300 medium-resolution displays are in a 12×15-pixel cell. These displays have no blinking mode (except for the alpha cursor), and no half-bright mode.

- High-resolution displays have 1 024 horizontal by 768[4] vertical pixels.

  Alpha capabilities of high-resolution displays are 48 lines of 128 characters, with no lines off-screen, like the Model 237. The characters are in an 8×16-pixel cell. These Series 300 high-resolution displays also have no half-bright mode and no blinking mode (except for the alpha cursor on all Series 300 displays except the 98700 display controller).

[1] There are three medium resolution displays (two monochrome and one color), and three high-resolution displays (one monochrome and two color).

[2] Series 300 medium-resolution displays actually have 1 024 horizontal pixels. However, BASIC graphics (but not alpha) handles contiguous pairs of horizontal (non-square) pixels as one unit in order to make square dots on the screen.

[3] Series 300 medium-resolution displays actually have 512 vertical pixels; however, only 400 are displayed.

[4] Series 300 high-resolution displays actually have 1 024 vertical pixels; however, only 768 are displayed.

## Processor Boards

Two processor boards are available with Series 300 computers:

- Medium-performance boards, which feature an MC68010 processor (10 MHz clock rate).
- Higher-performance boards, which feature an MC68020 processor (16 MHz clock rate) and an MC68881 floating-point math co-processor.

(Series 200 computers have either an MC68000 or MC68010 processor with an 8 or 12.5 MHz clock, depending on model numbers and product options.)

The 68010 is a 16-bit virtual memory microprocessor with a 32-bit internal architecture, while the MC68020 is a 32-bit microprocessor with an internal 256-byte instruction cache (which is normally operative but can be disabled by executing `CONTROL 32,3;0`).

The MC68020 also has a flexible co-processor interface that allows close coupling between the main processor and co-processors such as the MC68881 floating-point math co-processor. The MC68881, which provides full IEEE floating-point math support, can execute concurrently with the MC68020 and usually overlaps its processing with the 68020's processing to achieve higher performance. The MC68881 provides increased performance for floating-point operations, particularly for the evaluation of transcendental functions; refer to the "Efficient Use of the Computer's Resources" chapter for further details. (The MC68881 co-processor is normally operative, but you can disable it by executing `CONTROL 32,2;0`.)

## Battery-Backed Real-Time Clock

Series 300 computers have a built-in, battery-backed, real-time clock as well as a built-in volatile clock. Both have a lower limit of March 1, 1900. However, the upper limit of the volatile clock is August 4, 2079, while the upper limit of the non-volatile clock is February 29, 2000.

(Only Series 200 Models 226 and 236 could have optionally installed battery-backed, real-time clocks. This hardware was included with the HP 98270 Powerfail Option, whose main purpose was to provide power during brown-out or black-out situations.)

## Built-In Interfaces

All Series 300 computers have a built-in HP-IB interface, which is the same as the built-in HP-IB interface of all Series 200 computers.

Series 300 computers also feature the following built-in interfaces, which differ slightly from some of their Series 200 counterparts:

- RS-232C serial interface (like the HP 98644 low-cost serial interface).
- HP-HIL keyboard interface (like the one in Models 217 and 237)

### Serial Interface

All Series 300 computers have a built-in, 98644-like, serial interface. As with Series 200 Models 216 and 217 built-in serial interfaces, this interface is permanently set to select code 9. However, this interface differs slightly from versions of the Series 200 built-in serial interface (which are like the optional HP 98626 serial interface).

Since the goal of the 98644 is to provide a low-cost serial interface, there are no hardware switches that allow you to specify values for the following parameters:

- Select code (hard-wired to 9)
- Interrupt level (hard-wired to 5)
- Default baud rate (the BASIC system sets default to 9600 baud)
- Default line control parameters (the BASIC system sets defaults to 8 bits/character, 1 stop bit, parity disabled).

If your program expects any other values for the baud rate and line control parameters, you will have to change them programatically (select code and interrupt level cannot be set programmatically). See "Using a Configuration Program" in this chapter for further information.

## HP-HIL Keyboard Interface

Like the Series 200 Models 217 and 237 computers, Series 300 computers are equipped with ITF ("Integrated Terminal Family") keyboards[1] connected through an HP-HIL interface (Hewlett-Packard Human Interface Link).

---

**Note**

If you are porting existing Series 200 software to Series 300 and have already modified it to run on a Model 217 or 237 computer's ITF keyboard, then you have already made the adjustments necessary for this keyboard. If not, then continue reading this section.

---

The major human-interface differences between 98203 keyboards and ITF keyboards are in the number and layout of "user" and "system" function keys.

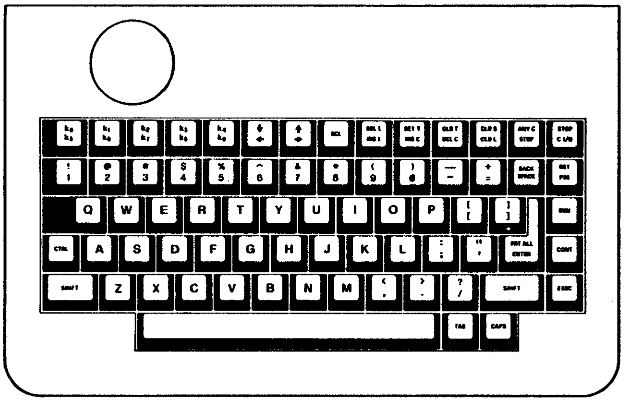


**Figure 15-1. HP 98203A Keyboard**

[1] With BASIC 4.0 and subsequent system revisions, the HP 98203C keyboard is also *optionally* available. This keyboard has the layout of the HP 98203B keyboard (and Model 226 and 236 built-in keyboards), but it is connected to the computer through the HP-HIL interface. If you will be using this keyboard, you will not have to make any changes to programs that use the HP 98203B-style keyboard.

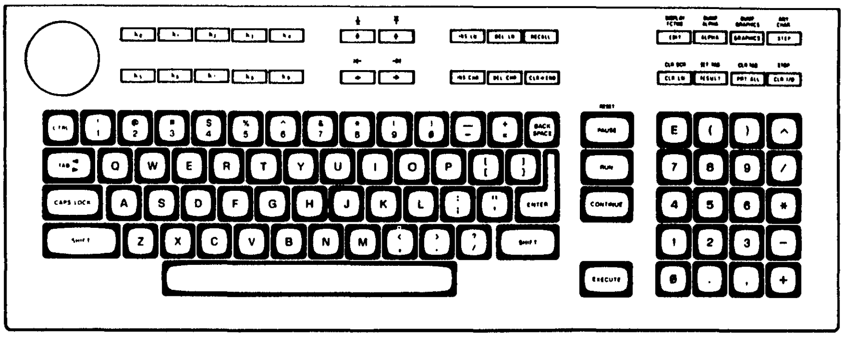


**Figure 15-2. HP 98203B/C Keyboard**

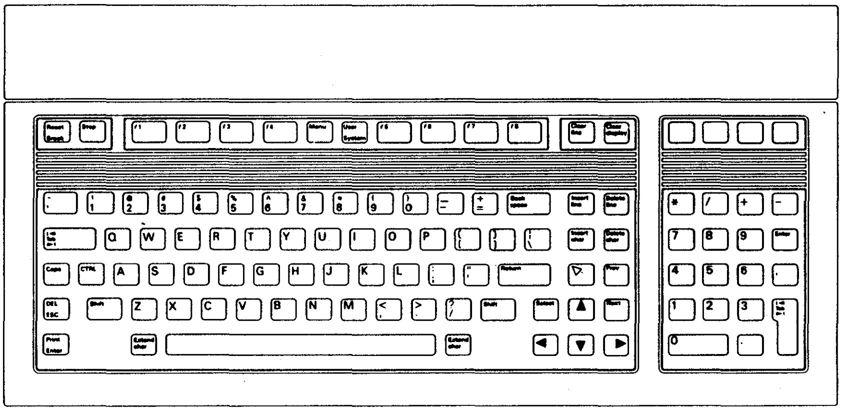


**Figure 15-3. ITF Keyboard**

Note that the ITF keyboard has only eight *physical* "user" function keys ($\boxed{\text{f1}}$ through $\boxed{\text{f8}}$, rather than $\boxed{\text{k0}}$ through $\boxed{\text{k9}}$), and lacks some of the *physical* "system" keys (such as $\boxed{\text{ALPHA}}$ and $\boxed{\text{RUN}}$). However, ITF keyboards actually have *more* functionality than 98203 keyboards, because BASIC provides several "system" and "user" definitions for ITF function keys $\boxed{\text{f1}}$ through $\boxed{\text{f8}}$. For complete definitions of each key on every keyboard, see the "Keyboard Reference" chapter of the *Using the BASIC System* manual.

BASIC also provides a way to emulate the operation of a 98203 keyboard using an ITF keyboard. Using this mode is a convenient way of porting Series 200 programs to Series 300 machines without modifying the source program. For further details of the "98203 compatibility mode", see the subsequent "Using a Configuration Program" section[1].

Also note that the 98203 keyboards can produce some keycodes that cannot be produced with the 46020 keyboard. These keycodes are produced by pressing the EXECUTE and EDIT keys. Thus if the Series 200 program depends upon these keycodes, the source code must be modified. See the subsequent "Modifying the Source Program" section for further details.

## ID PROM

Note that there is no built-in ID PROM available with Series 300 computers, as was the case with many models of Series 200 computers. However, an equivalent feature is provided by an optional HP-HIL device—the 46084A ID Module.

If the program reads the ID PROM's contents with a SYSTEM$("SERIAL NUMBER") function call, then the program will also read the ID Module's contents correctly. See "Software Security" in the "Entering, Running, and Storing Programs" chapter for further information. However, if its contents were read by a CSUB[1], then you will need to use a version that does not read the ID PROM.

[1] A keyboard overlay is provided with the system to label BASIC definitions of several ITF keys. The subsequent "98203 Keyboard Compatibility Mode" section describes the use of this overlay in both normal and compatibility modes.

[2] CSUB stands for Compiled SUBroutine, which is a program written in Pascal and generated using the CSUB Utility.

## Just Loading and Running Programs

This is the most desirable method, since it requires the least amount of work — just load the program into the Series 300 computer, and run it.

You can probably port most of your BASIC 3.0 or 3.01 programs this way.

There are three different actions you can take, depending on who developed your program:

- If HP developed the program, look in the "Operating Systems and Applications" section of the *HP 9000 Series 300 Configuration Reference Manual.* The manual shows which 3.0 or 3.01 applications will run on a Series 300 computer using the 4.0 system.
- If another software vendor developed the program, check with that vendor to determine whether it will run on a Series 300 computer. (You can also take one of the two actions listed below.)
- If you developed the program, you can do one of two things:
    - Read through the following sections to see whether it requires another porting method.
    - Try running it.

### Should Problems Arise

If your program will not run on your Series 300 system, then you may want to make considerations such as the following:

- Does it meet all of the criteria listed in the subsequent sections?
- Is there sufficient memory in the computer?
- Are all the necessary devices and corresponding device drivers installed?
- Have you fulfilled *all* other requirements listed by the software developer?

If the program still doesn't run, then you may want to call the organization responsible for supporting the program (the programmer, the software vendor, or HP).

## Using a Configuration Program

This method involves writing a program that configures the system for your program. Here are the situations for which this porting method will work:

- The program depends on a "non-default" 98626 serial interface configuration as set by hardware switches.
- The program depends on the 98203 keyboard layout (but does not depend on trapping the [EXECUTE] or [EDIT] keys).
- The program depends on separate alpha and graphics planes (and you have a Series 300 color display which you can configure to have separate alpha and graphics).

### HP 98644 Serial Interface Configuration

Here is an example situation for which you could use this method. Suppose your program depends on reading the following "non-default" parameters from the configuration switches on the 98626-like, built-in serial interface in a Model 217:

- 4800 baud
- 7 bits per character (with 1 stop bit) and odd parity.

However, the default parameters for the built-in 98644-like interface in Series 300 computers are as follows:

- 9600 baud
- 8 bits/character (with 1 stop bit), and parity disabled

One solution is to use a short program that selects the desired "non-default" baud rate (4800) and line-control parameters (7 bits, odd parity). This example program changes the "default" parameters by writing to CONTROL registers 13 and 14. (Note that you can also execute these CONTROL statements directly from the keyboard.)

```
100  CONTROL 9,13;4800 !                 Baud rate.
110  CONTROL 9,14;IVAL("11001010",2) !   No handshake (bits 7,6)
120  !                                   Odd parity   (bits 5-3)
130  !                                   1 stop bit   (bit  2)
140  END !                               7 bits/char  (bits 1,0)
```

Enter and run this program on the 4.0 system, making sure that the SERIAL binary program is installed beforehand. The serial card is properly configured by this program, which you may want to verify by reading the corresponding STATUS registers. You can then run the application program.

Another solution is to modify the source program to select these parameters (i.e., insert this segment of code into the program). In such case, you could change the "current" parameters by writing to CONTROL registers 3 (baud rate) and 4 (line control). However, if the interface is reset with the SCRATCH A statement, then the values in these registers will be restored to the "default" values currently in registers 13 and 14. See the *BASIC Interfacing Techniques* manual for details on the serial interface registers.

## HP 98203 Keyboard Compatibility Mode

The BASIC system provides a mode of keyboard operation in which the ITF keyboards are compatible with (i.e., emulate) 98203 keyboards. Before describing how the compatibility mode works, it will be helpful to review each keyboard's layout and normal operation.

### Brief Comparison of Keyboard Layouts

Below are diagrams of each keyboard. They are shown here for the purpose of comparing their physical differences. For a key-by-key description of each one, refer to the "Keyboard Reference" setion of the *Using the BASIC System* manual.

Here are the layouts of the 98203 keyboards:

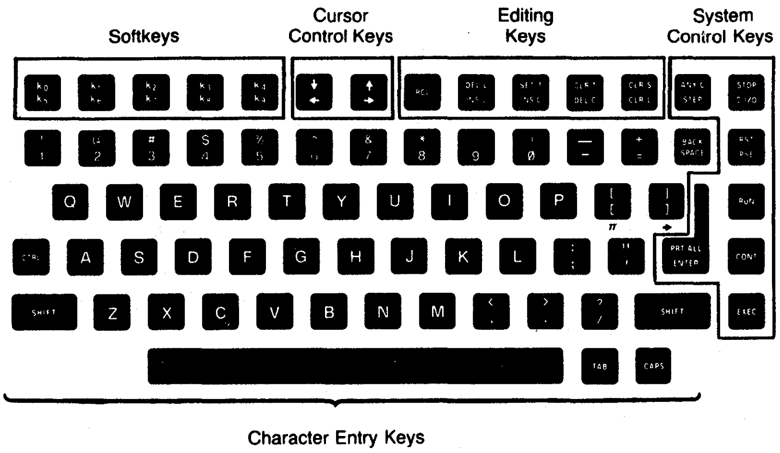


**Figure 15-4. HP 98203A Keyboard**

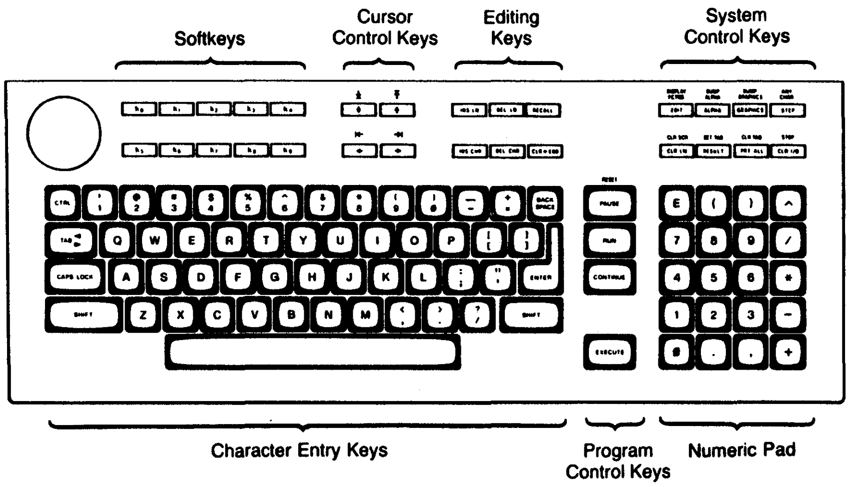


**Figure 15-5. HP 98203B and C Keyboards**

Note the "system" keys across the top of the keyboard (two rows across the top and one column down the middle of the larger 98203B; one row across the top and one column down the right side of the smaller 98203A).

Softkeys on the 98203 keyboards are labeled [k0] through [k9]. There are corresponding "softkey labels" which can be displayed on the alpha screen. For instance, you can enable the display of the default "typing-aid" labels by executing this statement:

```
LOAD BIN "KBD"
```

If this binary is already loaded and the "typing-aid" definitions are not currently displayed, execute LOAD KEY (with no file specifier).

Here is the format of the 98203 softkey labels. (Note that they match the physical layout of the softkeys.)

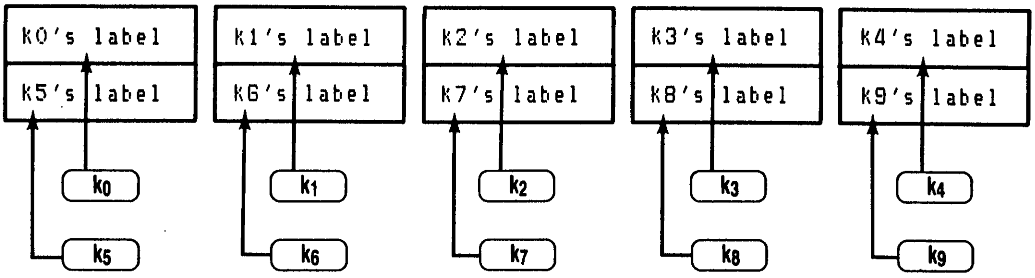


**Figure 15-6. HP 98203 Softkey Labels**

There are 2 rows of 5 labels each. Each label consists of up to 14 characters.

Contrast this layout to that of the ITF keyboards:

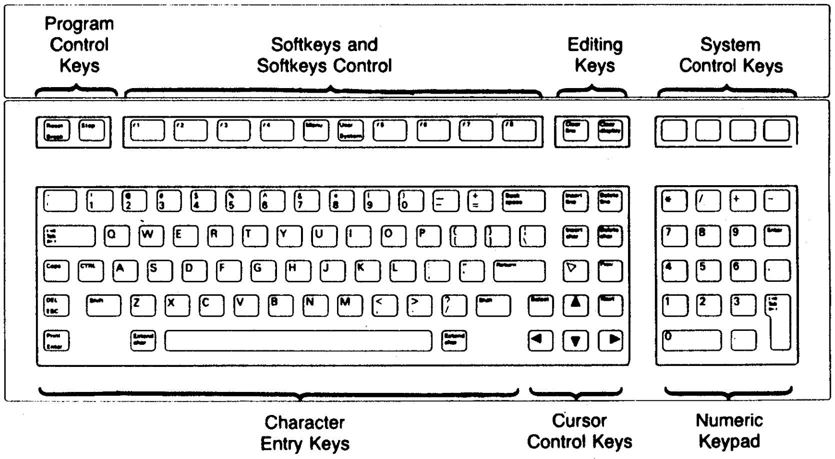


**Figure 15-7. ITF Keyboards (such as the 46020)**

Here are the default ITF "typing-aid" labels and corresponding keys. There is 1 row of 8 labels. Each label consists of up to 16 characters (2 rows of 8 characters per label).

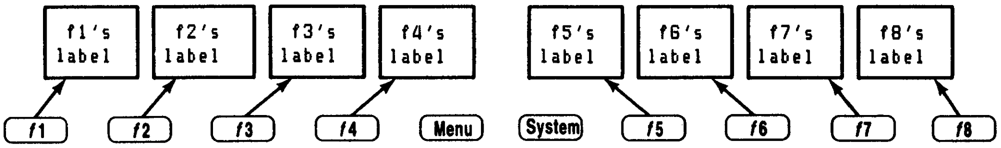


**Figure 15-8. HIL "Typing-Aid" Softkey Labels**

Even though the ITF keyboards have fewer physical function keys, they have *more* functionality than 98203 keyboards. This additional functionality is due to the fact that BASIC provides 1 menu of "system" keys (shown below) and 3 menus of "User" definitions for softkeys f1 through f8.

Here is the ITF "System" menu of keys, which you can display by pressing the Menu key (if labels are not already displayed) and then the System key:

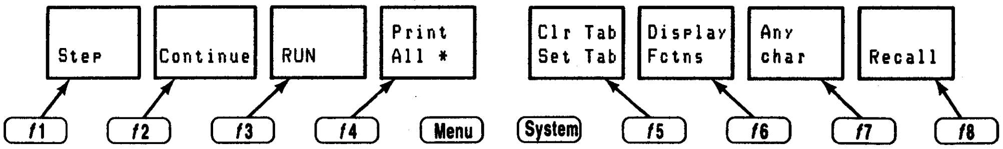


**Figure 15-9. HIL "System" Menu Labels**

This menu of softkey definitions provides most of the 98203 system key functions.

As you can see, there are two main areas of differences between 98203 keyboards and ITF keyboards:

- There are several "system" keys on the 98203 keyboards, such as STEP, CONTINUE (CONT on the smaller 98203A keyboard) and RECALL (RCL on the 98203A). These system functions are not written on the key-cap labels of ITF keyboards, but the BASIC system functions are available on the System menu.
- Softkeys on the 98203 keyboards are labeled k0 through k9. Thus, there are 20 softkeys available on the larger 98203 keyboards (by using SHIFT), and 10 on the smaller 98203 keyboard. Softkeys on the ITF keyboard are labeled f1 through f8. Thus, there are 24 softkeys available on these keyboards (3 menus of 8 keys each). The number and size of screen labels are also different.

## Enabling Keyboard Compatibility Mode

You can enter this mode by writing a non-zero value into keyboard control register 15:

```
CONTROL KBD,15;1
```

The following correspondence between function keys and labels is established[1]:

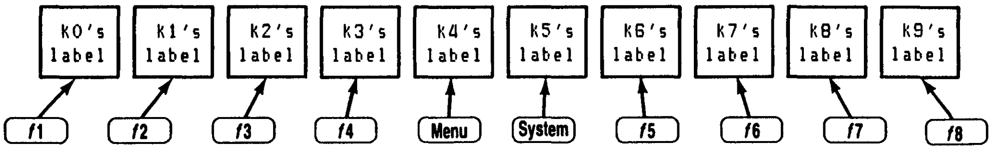


**Figure 15-10. Correspondence Between Function Keys and Labels**

There is 1 row of labels, and each label may have up to 14 characters (two rows of 7 characters each).

If you want to fully emulate the 98203 keyboard and corresponding softkeys' display behavior, you will need to execute the following statements:

```
CONTROL CRT,12;0
LOAD KEY
```

The CONTROL statement sets up the "key labels display mode" to match the default behavior of a display with the 98203 keyboard. The LOAD KEY statement loads the default "typing-aid" softkey definitions for the 98203 keyboards.

[1] If you are in edit mode when you enter this compatibility mode, then edit mode is canceled.

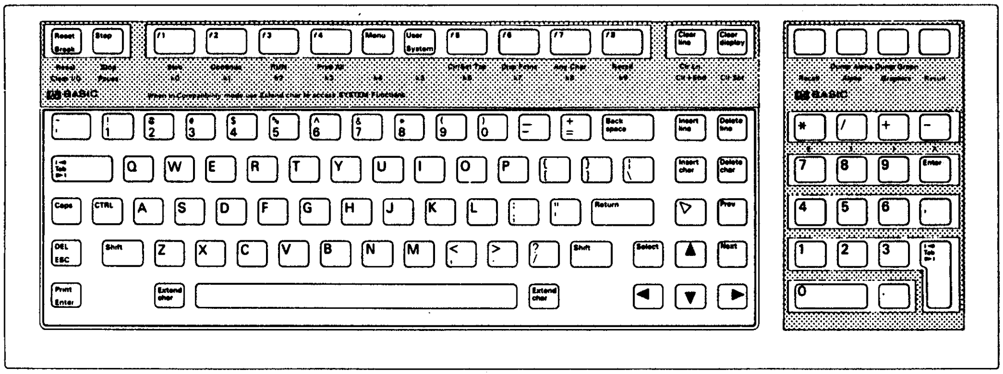
## Using Compatibility Mode

Here is a listing of the correspondence between ITF keys and 98203 keys while in this mode. For a detailed description of each 98203 key's function, see the "Keyboard Reference" chapter of *Using the BASIC System* manual.

---

**Note**

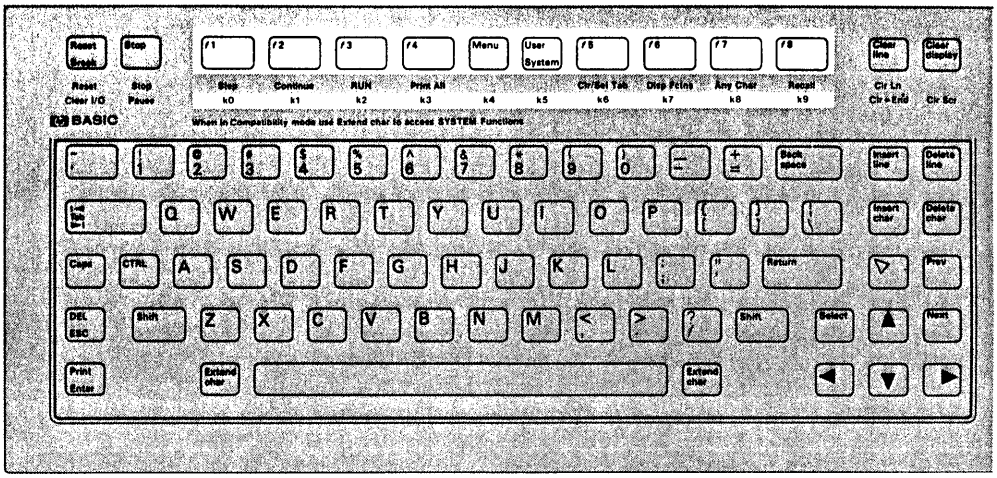
Place the BASIC keyboard overlays on the ITF keyboard before reading this section. Also note that you can use these overlays in normal mode as well as in compatibility mode.



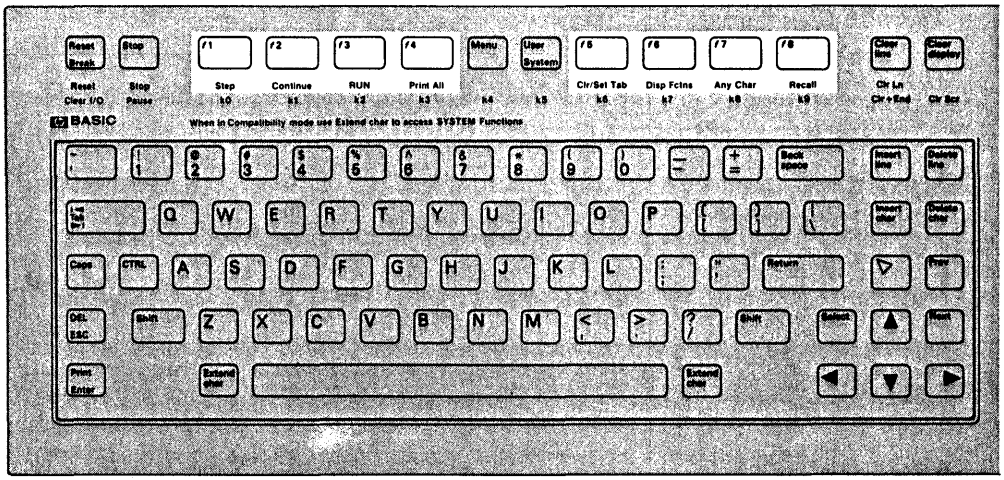
---

- To access a 98203 **softkey** definition, merely press the appropriate ITF softkey. For instance, the ITF [f1] softkey emulates the 98203 [k0] softkey, and the ITF [Menu] key emulates the 98203 [k4] softkey. (These key definitions are printed on the *bottom* row of the keyboard overlay.)

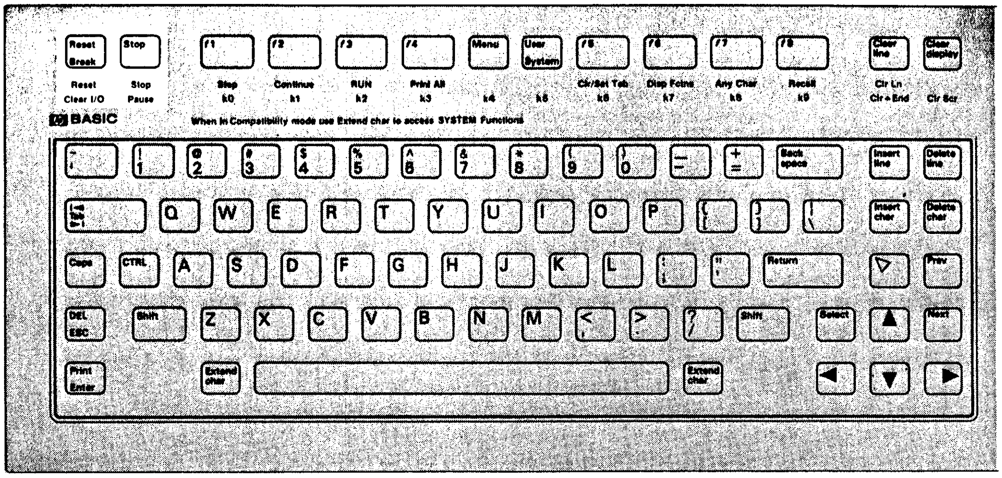
  Similarly, 98203 softkeys [k1] through [k1] are accessed by pressing the ITF [Shift] key with the appropriate softkey.



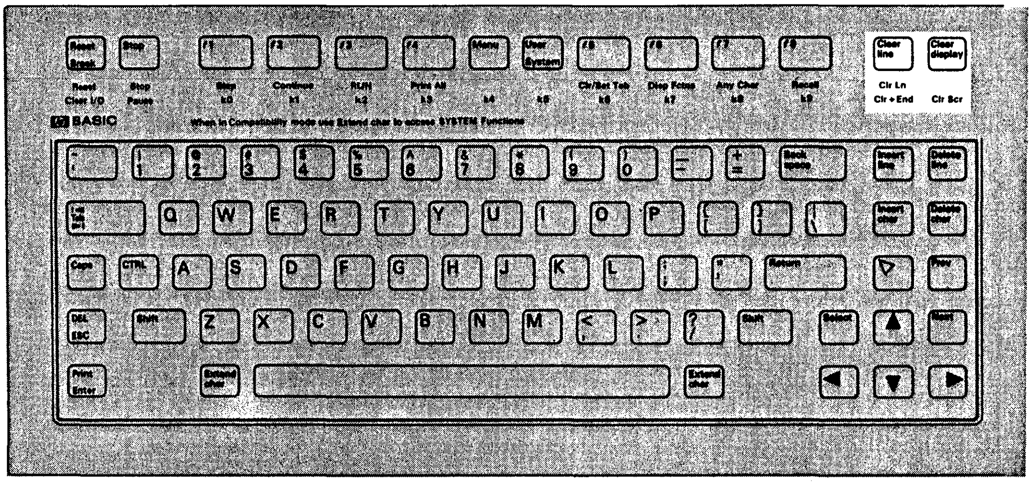
- To access a 98203 **system-key** definition, press [Extend char] with the appropriate ITF softkey. For instance, the ITF [Extend char]-[f1] key emulates the 98203 [STEP] key. (These key definitions are printed on the *top* row of the keyboard overlay. Note that these definitions are the same as in the normal-mode System softkey menu.)



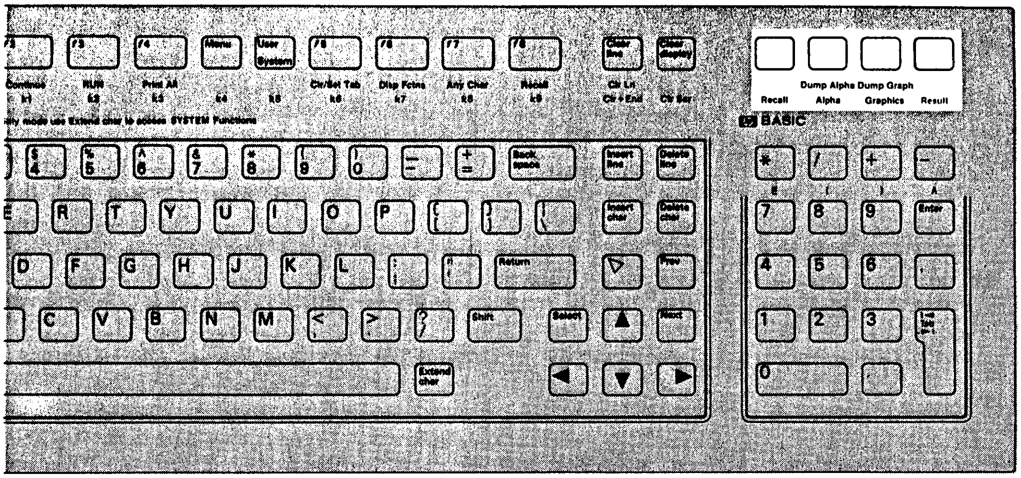
- The 98203 [CLR I/O] and [PAUSE] **system-key** definitions are available by using the ITF [Break] and [Stop] keys (*without* pressing [Extend char]). Note that these key definitions are the same in normal mode.



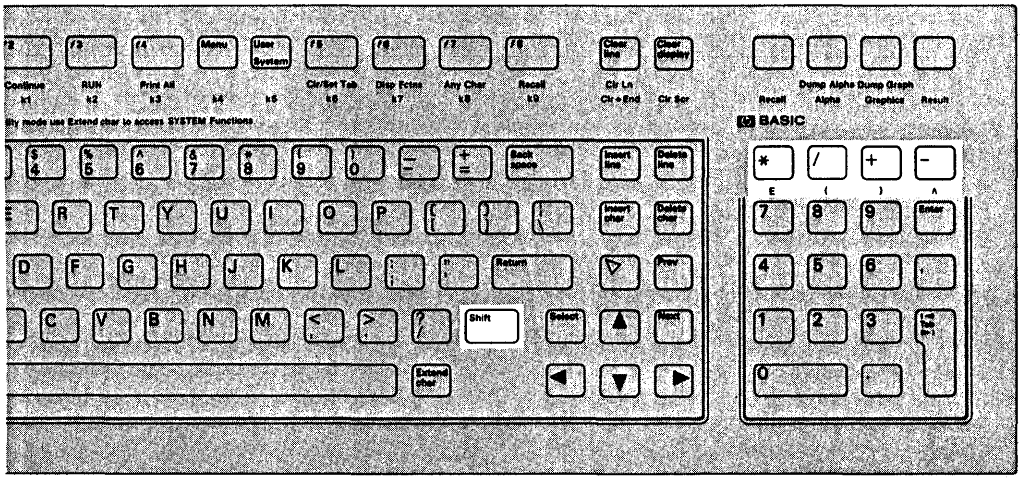
- The 98203 CLR→END, CLR LN, and CLR SCR **system-key** definitions are available by using the ITF Clear line, Shift Clear line, and Clear display keys. Note that these key definitions are the same in normal mode.



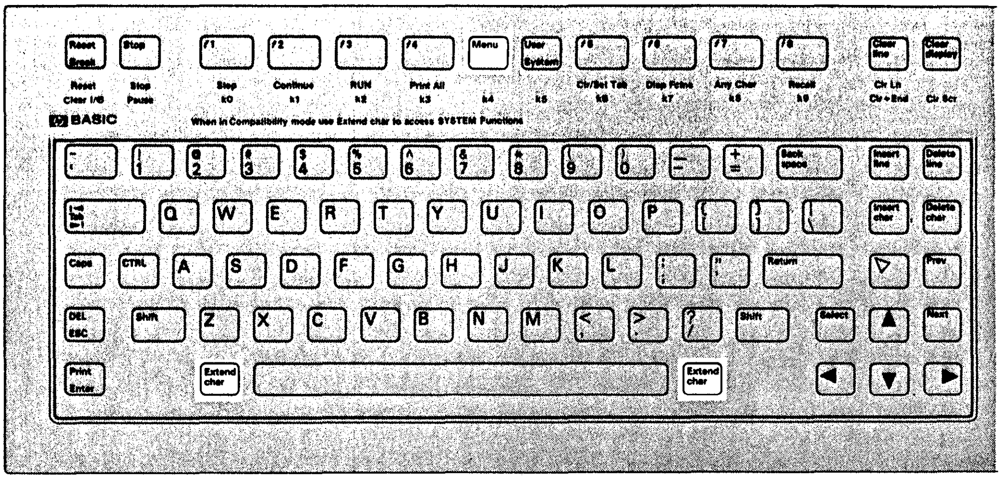
- The 98203 RECALL, ALPHA, GRAPHICS, and **RES system-key** definitions are available by using the *unlabeled* ITF keys above the numeric keypad. The shifted keys also have corresponding definitions (for example, Shift-**Alpha** is the DUMP ALPHA function). Note that these key definitions are the same in normal mode.



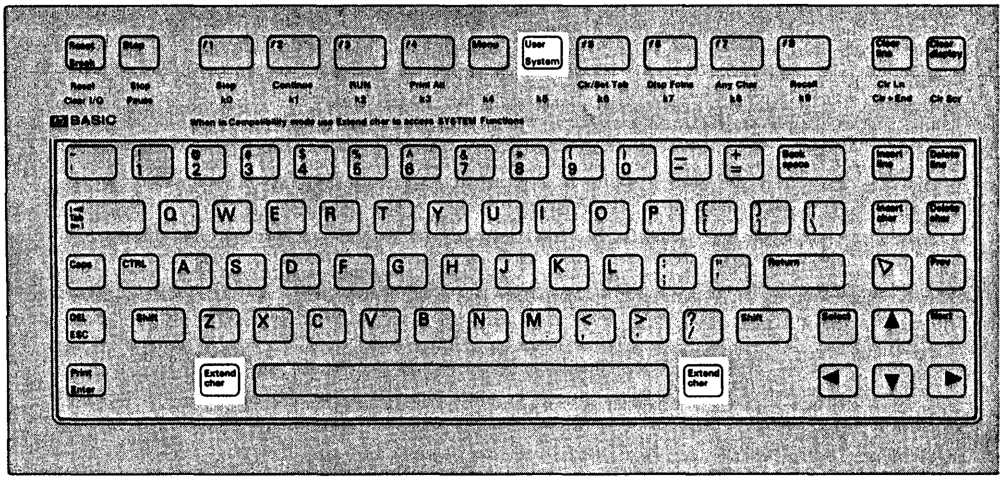
- When shifted, the $\boxed{*}$, $\boxed{/}$, $\boxed{+}$, and $\boxed{-}$ ITF keys on the top row of the numeric keypad have the same definitions as the keys on the top row of the 98203 numeric keypad. They are $\boxed{E}$ ($\boxed{\text{Shift}}$-$\boxed{*}$), $\boxed{(}$ ($\boxed{\text{Shift}}$-$\boxed{/}$), $\boxed{)}$ ($\boxed{\text{Shift}}$-$\boxed{+}$), and $\boxed{\wedge}$ ($\boxed{\text{Shift}}$-$\boxed{-}$). Note that these key definitions are the same in normal mode.



- Extend char-Menu is an on/off toggle for the **key labels**. (Extend char-Shift-Menu produces no visible change.)



- [Extend char]-[System] **exits compatibility mode**, and returns you to the ITF "System" key definitions. Similarly, [Extend char]-[User] exits this mode, and returns you to the ITF "User 1" key definitions. (Note that there is no corresponding keystroke to return to compatibility mode.)



## Exiting Keyboard Compatibility Mode

In addition to using the [Extend char]-[System] and [Extend char]-[User] keys to exit this mode, you can also use keyboard register 15:

```
CONTROL KBD,15;0
```

If the system is currently in edit mode, then exiting keyboard compatibility mode will also cancel the edit mode.

If you were emulating the 98203 keyboard and corresponding softkeys' display behavior (and want to return to the "normal" behavior), you will need to execute the following statements:

```
CONTROL CRT,12;2
LOAD KEY
```

The CONTROL statement restores the "key labels display mode" to the default behavior of a display with the ITF keyboard. The LOAD KEY statement restores the default "typing-aid" softkey definitions for the ITF keyboard.

## Configuring Separate Alpha and Graphics Planes

With BASIC 4.0 on bit-mapped color (multi-plane) displays, you have the ability to specify which planes are to be:

- write-enabled and used to display alpha
- write-enabled and used to display graphics

This feature allows you to simulate separate alpha and graphics of Series 200 displays. For instance, you will be able to:

- Turn alpha and graphics on and off independently.
- Dump them separately.
- Scroll alpha without scrolling graphics.

### An Example

Assuming that you have a four-plane display, you could enable plane 4 for alpha and planes 1 through 3 for graphics. The following program[1] performs this as well as some other operations, as described in the program's comments:

```
100   PLOTTER IS CRT,"INTERNAL";COLOR MAP ! Select Series 300 graphics.
110   FOR I=8 TO 15
120     SET PEN I INTENSITY 0,1,0         ! Set alpha pen colors (green).
130   NEXT I
140   CONTROL CRT,5;0                     ! Set alpha pen to black (temp.)
150   OUTPUT KBD;CHR$(255)&"K";           ! Clear alpha screen.
160   CONTROL CRT,18;8                    ! Select plane 4 for alpha.
170   CONTROL CRT,5;8                     ! Set alpha pen.
180   INTEGER Gm(0)                       ! Declare array for GESCAPE.
190   Gm(0)=7                             ! Set bits 2,1,0, which select
200   GESCAPE CRT,7,Gm(*)                 !  graphics planes 3,2,1.
210 ! PLOTTER IS CRT,"INTERNAL"            ! Return to non-color-map
220   END                                 !  mode (optional).
```

This program provides eight graphics pen colors (either the default or previously defined colors) and a single alpha pen color (green).

For more information concerning graphics displays, see the the "Multi-Plane Bit-Mapped Displays" section of the *BASIC Graphics Techniques* manual. For more information on alpha displays, see the "Display Interfaces" chapter of the *BASIC Interfacing Techniques* manual.

[1] Note that BASIC 5.0 provides the SEPARATE ALPHA FROM GRAPHICS statement to perform nearly the same functions as this program; see the *BASIC Language Reference* for details.

## Using the Display Compatibility Interface

This method involves installing an HP 98546 Display Compatibility Interface, which consists of essentially the *separate* graphics and alpha boards of the Series 200 Model 217 computer. You can then direct the system to use the compatibility display, enabling you to run existing Series 200 programs, which depend on this display's characteristics, on your Series 300 computer.

This card set remedies the following situations.

- The program depends on having separate alpha and graphics planes (and you do not have a color display which can emulate this feature, as described in the preceding "Configuring Separate Alpha and Graphics Planes" section).
- The program directly accesses alpha or graphics hardware (such as through a CSUB, rather than through a BASIC graphics statement).
- The program depends on blinking alpha display highlights (characters with codes 130, 134, and 135).
- The program depends on the Model 217's specific graphics resolution (512×390 pixels) or alpha display size (80×25 characters), or upon its specific alignment of graphics pixels and alpha pixels.

This method is required if any of the above statements is true **and** you cannot modify a program's source code (or don't want to). If you have the program's source code, then you may want to instead make the necessary modifications to it.

If your program requires separate alpha and graphics and also uses color, you have the option of using any color graphics display to drive a separate color monitor. Graphics can be displayed on this color monitor while alpha is display on the original monitor.

## Hardware Description

The card set consists of these two hardware pieces:

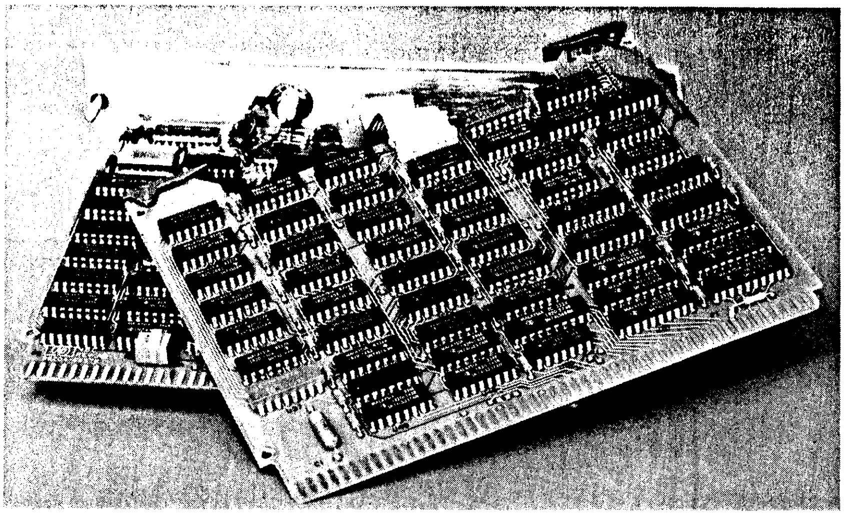


**Figure 15-11. The HP 98546 Display Compatibility Interface**

- An alpha display card, which is like the existing 98204B display controller card except for a relay and an additional BNC video connector on the rear panel.
- A graphics display card, which is identical to the Model 217's graphics card.

## The Relay and BNC Video Connectors

The relay on the alpha card is used to switch between using the Series 300 bit-mapped display's signal and using the compatibility display's signal.

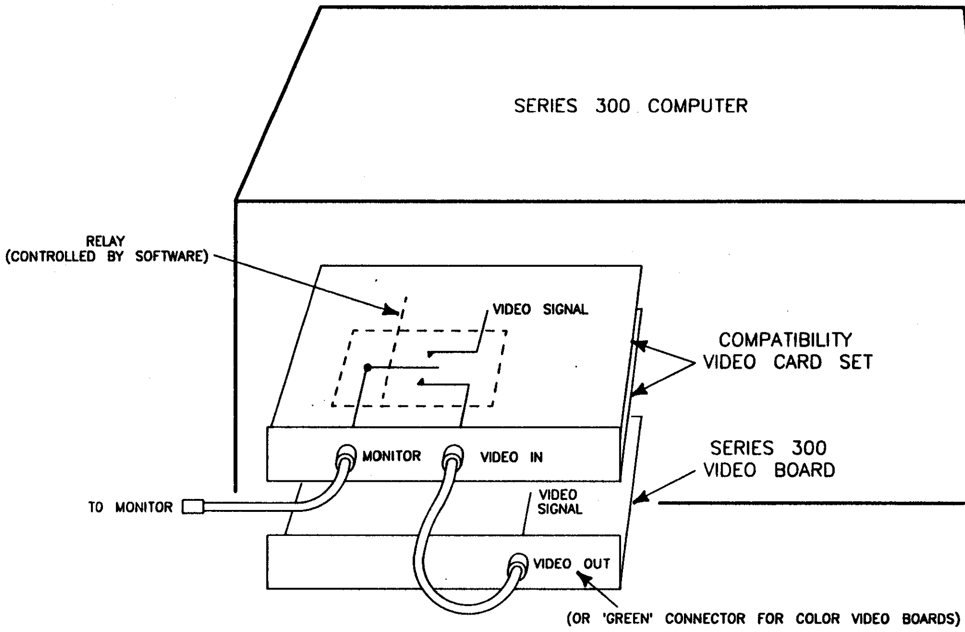

SERIES 300 COMPUTER
RELAY (CONTROLLED BY SOFTWARE)
VIDEO SIGNAL
COMPATIBILITY VIDEO CARD SET
MONITOR
VIDEO IN
SERIES 300 VIDEO BOARD
TO MONITOR
VIDEO SIGNAL
VIDEO OUT
(OR 'GREEN' CONNECTOR FOR COLOR VIDEO BOARDS)

**Figure 15-12. A Relay Governs Which Display Signal Is Used**

## Display Compatibility Interface Capabilities

Capabilities of this card are identical to those of the Model 217. The alpha display is an 80×25-character screen with half-bright, blinking, underline, and inverse-video display enhancements. The graphics display is 512×390 monochrome pixels.

### Configurations Possible

Here are the video-interface/monitor configurations possible:

- **Shared monitor:** The Display Compatibility Interface and the Series 300 bit-mapped display can share a medium-resolution monitor (monochrome or color).
- **Separate monitors:** The Display Compatibility Interface can use a medium-resolution monitor, and a Series 300 Video Board can use a separate monitor (monochrome or color, whichever is appropriate).
- **Single monitor:** The Display Compatibility Interface can use a medium-resolution monitor (with *no* Series 300 bit-mapped display).

## Steps in Using this Card Set

Here are the steps you will take with this method:

1. Turn off the computer.
2. Configure and install the Display Compatibility Interface according to the instructions in its *Installation Note.* Also connect the monitor(s) as described in that note.
3. Turn on the computer, and boot the BASIC system.
4. Load the CRTA display driver binary, if not already installed.

   ```
   LOAD BIN "CRTA" [Return]
   ```

5. Select the Display Compatibility Interface as the display device.

   ```
   CONTROL CRT,21;1 [Return]
   ```

**Note**

When using one monitor for two different displays (as in the "shared monitor" configuration described earlier), a small amount of time is required for the monitor to synchronize with the new display whenever you switch from one display to the other. Do not be disconcerted if the screen sometimes flickers when this switch is made.

The preceding CONTROL statement also performs the following actions:

- Chooses[1] and sets up the Display Compatibility Interface's alpha display as appropriate:
    - Sets all CRT registers to the appropriate default values.
    - Clears the Series 300 bit-mapped display screen.
    - Displays a cursor.
    - Displays key labels (if appropriate) in half-bright mode.
    - Displays a status indicator, such as the run light (if appropriate).
- Chooses[2] and sets up the Display Compatibility Interface's graphics display by effectively initializing this display and executing `GINIT` and `PLOTTER IS CRT,"INTERNAL"`.

## Switching Back to the Series 300 Display

The CONTROL statement is also used to select the Series 300 display:

```
CONTROL CRT,21;0 [Return]
```

The preceding CONTROL statement performs the following actions:

- Chooses[3] and sets up the Series 300's alpha display as appropriate:
    - Sets all CRT registers to the appropriate default values.
    - Clears the Display Compatibility Interface's alpha display.
    - Displays a cursor.
    - Displays key labels (if appropriate).
    - Displays a status indicator, such as the run light (if appropriate).
- Chooses[4] and sets up the Series 300 graphics display by effectively initializing the bit-mapped display and executing `GINIT` and `PLOTTER IS CRT,"INTERNAL"`.

[1] See "How the Default Alpha Display Is Chosen" in the "Display Interfaces" chapter of *BASIC Interfacing Techniques.* Items 1 and 2 are exchanged and a new selection of the "default display device" is made.

[2] The "default graphics display" is chosen according to the order listed under PLOTTER IS in the *BASIC Language Reference.*

[3] See "How the Default Alpha Display Is Chosen" in the "Display Interfaces" chapter of *BASIC Interfacing Techniques.* A new selection of the "default display device" is made. (Items 1 and 2 are **not** exchanged as in the switch to the Display Compatibility Interface.)

[4] The "default graphics display" is chosen according to the order listed under PLOTTER IS in the *BASIC Language Reference.*

## Automatic Display Selection at System Boot

When the BASIC system is booted with *both* the Display Compatibility Interface and the Series 300 bit-mapped display installed, it automatically selects one of them in the following manner:

- If only the CRTA driver is installed, the system selects the Display Compatibility Interface.
- If only the CRTB driver is installed (or if both CRTA and CRTB are present), the system selects the Series 300 bit-mapped display.

If only the Display Compatibility Interface is installed, the system selects it as the display (CRTA must be currently installed, of course). For a more detailed description of how the BASIC system selects the "default display device," see the "Display Interfaces" chapter of *BASIC Interfacing Techniques.*

## Removing Display Drivers

You can use SCRATCH BIN to remove all but the currently required display driver. In other words, if you are in compatibility display mode, then CRTB is removed. If you are in "native" Series 300 display mode (i.e., not in compatibility mode), then CRTA is removed.

## If Your Screen Is Blank

Your screen can go blank (and characters you type in from the keyboard are not "echoed" on the screen) under the following conditions:

- You have both a Display Compatibility Interface and a Series 300 bit-mapped display installed, and they are sharing the same monitor.
- You are not in compatibility mode (i.e., alpha is on the bit-mapped display).
- You are running a BASIC program that contains the following statement:

```
PLOTTER IS 3,"INTERNAL"
```

The execution of this statement causes your screen to go blank. You have just lost your alpha and graphics.

### What Happened?

The PLOTTER IS 3, "INTERNAL" statement changed the current plotter device from 6 (bit-mapped display) to 3 (compatibility display). The system is talking to the compatibility cards, and the software-controlled relay that switches from the bit-mapped to the compatibility display has been (implicitly) directed to switch to the compatibility display's video signal. However, the remainder of the operations performed by the CONTROL CRT,21;1 statement have not been performed. Therefore, you will not be able to see your alpha or graphics.

### What To Do Next

**Temporary solution:** You can do one of two things:

- To return to the bit-mapped display, first press the [Reset] key, and then execute a `SCRATCH A` or `CONTROL CRT,21;0` statement.
- To select the Display Compatibility Interface, execute `CONTROL CRT,21;1`.

Note that you will not see any characters echoed on the display until you have executed one of the above statements.

**Long-term solution:** Change all references to select code "3" to "CRT" (e.g. `PLOTTER IS CRT,"INTERNAL"`).

### Another Related Note

If you want to determine how well your program runs on a Series 300 bit-mapped display and this program executes a PLOTTER IS 3, "INTERNAL" statement, and you have Display Compatibility Interface installed, then you will not be able to adequately test the functionality of your software on a bit-mapped display unless you first remove the compatibility hardware (or change the PLOTTER IS 3, "INTERNAL" statements to PLOTTER IS CRT,"INTERNAL").

## Modifying the Source Program (Porting to 4.0)

This method involves changing or adding to the program's source code to make an existing (pre-4.0) program perform the desired operations on the 4.0 system.

Here are some, but not all, situations for which this method is required:

- The program depends on a CSUB with version 3.01 (or earlier).
- The program depends upon trapping HP 98203 EXECUTE or EDIT key codes, which cannot be generated by an ITF keyboard.
- None of the preceding porting methods worked. (In such case, you should read the subsequent "Additional Porting Considerations" section to see if your problem is described therein.)

If any of the above statements is true, then you must modify the program to run on the 4.0 system. If you do **not** have access to the source code, then you **cannot** port it—you will have to obtain a BASIC 4.0 version of the program, if it is available.

### Incompatible CSUBs

An example of this situation is a program that depends upon using a "pre-4.0" CSUB.

To remedy this situation, you will need to obtain a CSUB that is compatible with the BASIC 4.0 system. (This may require modifying the CSUB source program; it will definitely require re-generating a new CSUB with the CSUB *4.0* Utility.)

### HP 98203 Specific Key Codes

The 98203 keyboards can generate EXECUTE and EDIT key codes which cannot be generated by a 46020 keyboard. If your program depends on trapping these key codes, then you will need to modify it to use 46020 keys instead. For instance, you could trap the ITF Select key rather than the 98203 EXECUTE key. See the "Keyboard Interfaces" chapter of the *BASIC Interfacing Techniques* manual for examples of trapping keystrokes with a BASIC program.

## Additional Porting Considerations

This section describes the following topics, which may also require consideration in porting programs from "pre-4.0" BASIC programs to the BASIC 4.0 system.

- New SYSTEM$("SYSTEM ID") values for Series 300 computers
- Alpha color changes on Series 300 color displays
- Alpha screen height and graphics scrolling
- GLOAD/GSTORE compatibility
- PLOTTER IS statement
- Hidden color changes
- ON KNOB "interval" parameter for HP-HIL knobs

### New SYSTEM$("SYSTEM ID") Values

On Series 300 computers, SYSTEM$("SYSTEM ID") will return two different values:

- `S300:10` for computers with an MC68010 processor
- `S300:20` for computers with an MC68020 processor

### Alpha Color Changes

With multi-plane bit-mapped displays, printing one of the alpha color highlight characters, CHR$(136) through CHR$(143), will provide the same colors as on the Model 236C as long as the color map contains *default* values. A user-defined color map which changes the values of any pen in the range 0 to 7 will consequently change the effect of the corresponding color highlight character. See "Display-Enhancement Characters" in the "Useful Tables" appendix of the *BASIC Language Reference* for more information.

### Alpha Screen Height and Graphics Scrolling

With BASIC 3.0 and later versions, you can limit the height of the alpha portion of the screen. For instance, to limit the alpha portion of the screen to the bottom 11 lines, execute this statement:

```
CONTROL CRT,13;11
```

The screen height parameter of 11 specifies the number of lines to be used for the alpha screen (4 lines of "output area," and 7 lines used by the system). The value of this parameter may not be less than 9. A corresponding STATUS statement will return the current screen height.

This capability allows you to separate alpha and graphics on a single-plane bit-mapped display screen. You would also have to limit graphics to the upper portion of the screen (which is not used for alpha).

### GLOAD/GSTORE Compatibility

Raster images loaded by GLOAD should have been stored (GSTORE) from the same type of display. Otherwise, if the image was stored on a machine with a different graphics resolution or number of bits per pixel, then the resultant image will be scrambled.

If your program first creates a graphics image and then GSTOREs and GLOADs it, then the image may be truncated (due to the difference in required array sizes). With BASIC 4.0, you can use the GESCAPE statement to determine the required array size.

For example, the Model 236C requires an integer array size of 49 920 elements to store information from the graphics planes in the frame buffer [(4 bits/pixel)×(512×390 pixels)/(16 bits/integer)], while a Series 300 medium-resolution color display requires 102 400 elements (4×(1024×400)/16]). The value of 1024 is used because Series 300 medium-resolution bit-mapped displays have non-square-pixels.

See GLOAD and GSTORE in the *BASIC Language Reference* for details concerning this topic. With BASIC 4.0, there are new utility CSUBs (Bstore and Bload) that allow you to store and load *specified portions* of the graphics raster. You may alternatively want to use these utilities in favor of using GSTORE and GLOAD.

### PLOTTER IS Changes

There are several values that you can use when specifying the graphics display; however, the following examples show the best way:

```
PLOTTER IS CRT,"INTERNAL"
PLOTTER IS 1,"INTERNAL"
```

CRT is a built-in function that always returns 1. The value of 1 signifies the "default display" (to the PLOTTER IS statement).

The following statement, with select code of 3, specifies a non-bit-mapped display, if there is one; otherwise it is the same as PLOTTER IS 1,"INTERNAL".

```
PLOTTER IS 3,"INTERNAL"
```

The following statement *always* specifies a bit-mapped display. If one is not currently installed, then an error results.

```
PLOTTER IS 6,"INTERNAL"
```

Refer to the *BASIC Language Reference* for further details on the PLOTTER IS statement.

### Hidden Color Changes

On a Model 236C display, the following sequence of commands:

```
GRAPHICS OFF
SET PEN 0 INTENSITY 1,0,1
GRAPHICS ON
```

produces the following results.

- The GRAPHICS OFF statement will turn the graphics display off.
- SET PEN 0 is executed while the graphics screen is still blank and when the GRAPHICS ON statement is executed, the previous display contents with modified color map entry 0 is displayed.

On the Series 300 and 98700 displays, the above command sequence produces the following results:

- If the alpha and graphics planes overlap (i.e. the default configuration), then GRAPHICS OFF and GRAPHICS ON are no-op's, so the display will change immediately.
- If the alpha and graphics planes are totally independent (such as in "Configuring Separate Alpha and Graphics Planes" in the "Using a Configuration Program" section), then:
    - GRAPHICS OFF turns the graphics planes off, leaving the alpha plane on.
    - SET PEN n INTENSITY a,b,c will not be seen on the screen until the GRAPHICS ON statement is executed, *unless* n is equal to 0 or specifies an alpha pen.
    - GRAPHICS ON turns on the graphics planes again.

**Note**

This occurs because alpha and graphics share the same color map on Series 300 and 98700 displays, and PEN 0 is the default alpha background color.

## HP-HIL Knob Interval Parameter

The ON KNOB "interval" parameter for the optional HP-HIL knob (46083A) has been implemented in BASIC 4.0 (it was not implemented with HIL knobs in BASIC 3.0 or 3.01). This parameter works same way on an HIL knob as on the non-HIL knob (built into Series 200 98203 keyboards). See the "Using the Knob" section of the "Communicating with the Operator" chapter of this manual.

## BASIC 4.0 Enhancements for Series 200 Computers

Although the main objective of BASIC 4.0 was to add support of Series 300 computers, it also added some additional features for Series 200 computer users[1]. This section describes these enhancements.

**Table 15-1. BASIC 4.0 Enhancements for Series 200 Computers**

| Hardware Enhancements | Software Enhancements |
|---|---|
| **Support New HP-HIL Graphics Devices[2]:**<br>Tablets: HP 46087A (A size)<br>HP 46088A (B size)<br>TouchScreen: HP 35723 | (Still use GRAPHICS INPUT IS, DIGITIZE, READ LOCATOR, etc.) Can determine maximum hard clip values with GESCAPE operation selectors 20 through 22. (See the "Interactive Graphics" chapter of *BASIC Graphics Techniques.*) |
| **Ability to Specify Different Colors for Alpha Display Regions:**<br>Model 236C Only. | CRT STATUS/CONTROL registers 5 (modified definition) and 15 through 17 (new). (See the *BASIC Language Reference.*) |
| **New Graphics Utilities:**<br>No additional hardware is required. | "Bstore" and "Bload" utilities allow you to store and load specified portions of graphics rasters. "Gdump_rotated" allows you to dump graphics rotated by 90°. (See the "BASIC Utilities Library" chapter of the *Installing and Maintaining the BASIC System* manual.) |
| **HP 98644A Serial Interface Registers:**<br>Less-expensive than HP 98626A (but has fewer "default" configuration switches). | Interface STATUS/CONTROL registers 13 and 14 allow you to read and change the "SCRATCH A defaults" to get the functionality of switches. (See the "Serial Interface" chapter of *BASIC Interfacing Techniques.*) |

[1] These enhancements also pertain to Series 300 computers.
[2] These devices can only be connected to computers with an HP-HIL interface. For Series 200 computers, it includes Model 217 and Model 237 computers, and Model 220 computers with an optional HP-HIL interface.

**Table 15-1. BASIC 4.0 Enhancements for Series 200 Computers (Continued)**

| Hardware Enhancements | Software Enhancements |
|---|---|
| **HP 98203 Keyboard Compatibility Mode:**<br><br>None (useful with Models 217 and 237; also with 220 that uses the optional ITF keyboard). | KBD CONTROL register 15 enables the ITF keyboard to emulate the HP 98203B (Model 226/236) keyboard. (See the preceding "HP 98203 Keyboard Compatibility Mode" section of this chapter.) |
| **Support New Foreign-Language ITF Keyboards**[1]**:**<br><br>Revised HIL "Swiss French*" and "Swiss German*" keyboards are now supported. | SYSTEM$("KEYBOARD LANGUAGE") returns corresponding identifier. (See the *BASIC Language Reference.*) |
| **HPHIL Knob Interval Parameter**[1]**:**<br><br>None (same HIL knob as before). | With BASIC 3.0, the interval parameter for ON KNOB was ignored for HIL knobs. With 4.0, the parameter is used. (See the *BASIC Language Reference.*) |
| **Read "Keyboard Input" Line (Non-Destructively):**<br><br>None. | Use SYSTEM$("KBD LINE"). (See the "Communicating with the Operator" chapter of this manual.) |

[1] These devices can only be connected to computers with an HP-HIL interface. For Series 200 computers, it includes Model 217 and Model 237 computers, and Model 220 computers with an optional HP-HIL interface.

# Porting to 5.0 16

# Porting to 5.0 16

The BASIC 5.0 System is the latest revision of the "Series 200/300 Workstation BASIC" product. It consists of miscellaneous new features which further enhance the capabilities of this language and operating system. This chapter describes the incremental features[1] of BASIC 5.0, as well as describes the small changes made to some existing BASIC features. It will help you determine what to do when moving from the 4.0 to the 5.0 revision of this system.

## Compatibility with Previous Versions

As with most other version changes to this BASIC language, the 5.0 revision is highly compatible with preceding versions. In other words, using the BASIC 5.0 system:

- You can LOAD and RUN program (PROG) files created with STORE on previous versions of BASIC.
- You can GET and RUN program (ASCII) files created with SAVE on previous versions of BASIC.
- You can use all data files (BDAT and ASCII) created on previous versions of BASIC. (If you will be using the HFS directory format, then you should also read the "Porting and Sharing Files" chapter.)
- If you are using Compiled Subprograms (CSUB's), you will have to regenerate them using the CSUB 5.0 Utility.
- The BASIC editor is now in a separate binary (EDIT). If you require this, be sure to LOAD BIN "EDIT" before attempting to EDIT, LIST, or SAVE a program.
- The typing-aid softkey definitions have changed slightly from 4.0. If your application depends upon a particular typing-aid definition, then check to see whether it has changed. (If it has, then you can programmatically re-define it with LOAD KEY or SET KEY.)

[1] You may also want to note that the "Language History" section of the *BASIC Language Reference* provides a list of all keywords, indicating which ones were introduced in BASIC 5.0

## Categories of New Features

This section describes the general categories of 5.0 features. They are presented roughly in the order you would encounter them while using the system. (Subsequent sections further describe each category, and list where they are described in the BASIC manuals.)

- New hardware supported
- New utilities
- Hierarchical File System (HFS) support
- Human interface enhancements
- Keywords that duplicate register operations
- General programming additions.
- New STATUS and CONTROL registers
- Additional support for HP-HIL devices
- Additional graphics capabilities
- Additional CSUB capabilities

## New Hardware Supported

BASIC 5.0 is supported with the new Model 330 and 350 computers.

**Table 16-1. New Hardware Supported**

| Computer Model | BASIC Language Support |
|---|---|
| Model 330[1] | Supported in "Main" system (no binary is required). |
| Model 350[1] | Supported in "Main" system (no binary is required). |

[1] Note that the Local Area Network (LAN) interface available with some of these models is not supported by BASIC.

## New Utilities

The following utilities have been added to BASIC to simplify and speed up the installation, configuration, verification, and maintenance tasks.

**Table 16-2. New Utilities**

| New Feature | New Utility | Tutorial Information |
|---|---|---|
| Can verify the operation of discs, printers, plotters, HP-HIL devices, and HP-IB graphics tablets. Also helps you to label your mass storage devices, printers, and plotters. | Peripheral Verification Utility (VERIFY) | "Verifying and Labeling Peripheral" chapter of *Installing and Maintaining the BASIC System* |
| Can install BASIC on LIF and HFS hard and flexible discs, including formatting the disc[1] and storing the BASIC system on the disc. | System Disc Utility (DISC_UTIL) | "Putting BASIC on a Hard Disc" chapter of *Installing and Maintaining the BASIC System* |
| Can back up and restore entire disc and tape volumes, as well as individual files. Has the ability to specify files and directories with *wildcards.* (Uses the HP-UX *cpio* format.) | Backup Utility (BACKUP). | "Maintaining" section of *Installing and Maintaining the BASIC System* |
| Can edit the display font in bit-mapped alpha displays, store the new font in a file, and load it at a later time. | Font Editor Utility (FONT_ED) | "BASIC Utilities Library" chapter of *Installing and Maintaining the BASIC System* |
| Text editor and file-copy utilities called by pressing typing-aid softkeys. | Memory-Resident Utilities (MEM_UTILS) | "BASIC Utilities Library" chapter of *Installing and Maintaining the BASIC System* |

[1] The System Disc Utility is also used to format and check the consistency of an HFS disc, as described in the next section.

## HFS Disc Support

The following features have been added to BASIC (with the HFS binary) to support the Hierarchical File System (HFS) format for discs and other mass storage devices. This file system is compatible with Series 200/300 HP-UX (5.0 and later versions).

**Table 16-3. HFS Disc Support**

| New Feature | Supporting System Component | Tutorial Information |
|---|---|---|
| BASIC, HP-UX, and Pascal can reside on the same mass storage volume | System Disc Utility (DISC_UTIL) formats HFS volumes | "Putting BASIC on a Hard Disc" chapter of *Installing and Maintaining the BASIC System* |
| BASIC, HP-UX, and Pascal can access compatible files (ASCII and HP-UX). | CREATE and ASSIGN[1] | "Porting and Sharing Files" chapter of this manual |
| Hierarchical directories are supported (on both hard and flexible discs). | HFS-formatted volumes | "Using Directories and Files" chapter of *Using BASIC* |
| Extensible files are available | HFS files | "Data Storage and Retrieval" chapter in volume 1 of this manual |
| Access to the HP-UX file-protection scheme (on HFS directories) | PERMIT, CHGRP, and CHOWN statements | "Using Directories and Files" chapter of *Using BASIC* |
| Ability to detect and correct HFS inconsistencies | System Disc Utility (DISC_UTIL) | "Maintaining" section of *Installing and Maintaining the BASIC System* |

Note that time stamps are placed on HFS files whenever the BASIC system modifies the contents of the file. LIF files are also time-stamped with the BASIC 5.0 revision (previous versions did not do this).

[1] These are just a few of the I/O Operations (i.e. ENTER, OUTPUT, GET, etc. may also be included).

## Human Interface Enhancements

The following features have been added to the system to improve the BASIC system's human interface[1].

**Table 16-4. Human Interface Enhancements**

| New Feature | Supporting System Component (Binary Required) | Tutorial Information |
|---|---|---|
| New textual "run light" on the screen (systems with ITF keyboards only). | Enabled whenever softkey labels are on (No binary required) | "Loading and Running Programs" chapter of *Using BASIC* |
| Can clear the RECALL key buffer. | SCRATCH R (No binary required) | "Introduction to the System" chapter of *Using BASIC* |
| New default typing-aid key definitions. | No new keywords (No binary required) | Various locations in *Using BASIC* |
| Additional "sound" capabilities (on computers with HP-HIL interfaces) | SOUND (KBD binary) | "Communicating with the Operator" chapter in volume 1 of this manual |
| Redefinable character fonts (on bit-mapped alpha displays only). | SET CHR, CHRX, and CHRY (CRTX binary) | "Communicating with the Operator" chapter in volume 1 of this manual |
| Separated Program Editor/Lister (LIST) from main system | EDIT, LIST, and SAVE moved to EDIT binary | "Language Extensions, Drivers, and Configuration" chapter of *Installing and Maintaining the BASIC System* |
| New BASIC statements to clear display regions (formerly performed with OUTPUT KBD) | CLEAR SCREEN, CLEAR LINE (CRTX binary) | "Communicating with the Operator" chapter in volume 1 of this manual |
| Can load individual (or all) typing-aid softkeys programmatically | SET KEY (KBD binary) | "Communicating with the Operator" chapter in volume 1 of this manual |

[1] The editor and lister were put into the EDIT binary so that the entire "main" system could fit on a single disc, not to "improve" the human interface. It does, however, allow you to have a "run-only" system which might be useful in some applications.

## New Keywords that Duplicate Register Operations

Several STATUS and CONTROL register operations have been duplicated by keywords which perform the same action.

**Table 16-5. Keywords Duplicating Register Operations**

| New Keyword | Register Operation Duplicated |
|---|---|
| `DISPLAY FUNCTIONS ON` | `CONTROL CRT,4;1` |
| `DISPLAY FUNCTIONS OFF` | `CONTROL CRT,4;0` |
| `ALPHA PEN Pen_number` | `CONTROL CRT,5;Pen_number` |
| `KEY LABELS ON` | `CONTROL CRT,12;2` |
| `KEY LABELS OFF` | `CONTROL CRT,12;1` |
| `ALPHA HEIGHT Lines` | `CONTROL CRT,13;Lines` |
| `ALPHA HEIGHT` | Restores default (when `Lines` omitted) |
| `PRINT PEN Pen_number` | `CONTROL CRT,15;Pen_number` |
| `KEY LABELS PEN Pen_number` | `CONTROL CRT,16;Pen_number` |
| `KBD LINE PEN Pen_number` | `CONTROL CRT,17;Pen_number` |
| `SET ALPHA MASK Mask_value` | `CONTROL CRT,18;Mask_value` |
| `SET DISPLAY MASK Mask_value` | `CONTROL CRT,20;Mask_value` |
| `SYSTEM KEYS` | `CONTROL KBD,2;0` |
| `USER 1 KEYS` | `CONTROL KBD,2;1` |
| `USER 2 KEYS` | `CONTROL KBD,2;2` |
| `USER 3 KEYS` | `CONTROL KBD,2;3` |
| `KBD CMODE ON` | `CONTROL KBD,15;1` |
| `KBD CMODE OFF` | `CONTROL KBD,15;0` |

For tutorial information, see the "Display Interfaces" and "Keyboard Interfaces" chapters of *BASIC Interfacing Techniques.* (The KBD register statements are in the KBD binary; all others are in the CRTX binary.)

## General Programming Additions

The following features are used in BASIC programming.

**Table 16-6. General Programming Additions**

| New Feature | New Keyword (Binary Required) | Tutorial Information |
|---|---|---|
| Complex math | COMPLEX data type, supported in most math operations (COMPLEX binary) | "Numeric Computation" chapter in volume 1 of this manual |
| Hyperbolic functions | SINH, COSH, TANH, etc. (COMPLEX binary) | "Numeric Computation" chapter in volume 1 of this manual |
| Searching arrays for patterns and conditions | MAT SEARCH (MAT binary) | "Numeric Arrays" chapter in volume 1 of this manual |
| Copying subarrays | MAT enhancement (MAT binary) | "Numeric Arrays" chapter in volume 1 of this manual |
| New string variable function (returns DIMensioned string length). | MAXLEN function (No binary required) | "String Manipulation" chapter in volume 1 of this manual |
| Error-trapping feature enhancements. | CAUSE ERROR, ERRLN, ERROR RETURN, ERROR SUBEXIT, CLEAR ERROR (No binary required) | "Handling Errors" chapter in volume 1 of this manual |
| Can programmatically specify which system to re-boot | SYSBOOT enhancement (No binary required) | *BASIC Language Reference* |

## New STATUS/CONTROL Registers

The following new STATUS and CONTROL registers have been added in BASIC 5.0.

**Table 16-7. New STATUS/CONTROL Registers**

| New Register | Definition |
|---|---|
| `STATUS 32,4;Batt_clock_type` | Returns the following values:<br>0 ⇒ no battery-backed clock;<br>1 ⇒ HP 98270 battery-backed clock (Models 226 and 236 only);<br>2 ⇒ HP-HIL battery-backed clock. |
| `STATUS KBD,16 ;Scroll_disabled` | Reading the STATUS register allows you to determine whether the PRINT area of the display can be scrolled by keystrokes or equivalent operations (the default is to allow scrolling).<br>0 ⇒ scrolling enabled<br>1 ⇒ scrolling disabled |
| `CONTROL KBD,16; Disable_scroll` | Writing a 1 to the CONTROL register disables scrolling (useful to prevent scrolling of alpha display; writing a 0 to the register enables scrolling). |
| STATUS KBD,17;Auto_menu | Automatic menu switching:<br>1 ⇒ enable (default)<br>0 ⇒ disable |
| CONTROL KBD,17;Disable_auto | Automatic menu switching: mode.<br><>0 ⇒ enable<br>0 ⇒ disable<br>This register controls whether a system with an ITF keyboard will switch to (from) the User 2 Menu automatically on entering (leaving) EDIT mode. |

See the "Clock and Timers" chapter of this manual for details on determining clock type. See the "Keyboard Interfaces" chapter of  for details of disabling scrolling. Also see the descriptions of these registers in the "Useful Tables" section of this manual or the *BASIC Language Reference*.

## Additional HP-HIL Support

The following features provide greater support for Hewlett-Packard Human Interface Link (HP-HIL) devices. All of these capabilities require the KBD binary.

**Table 16-8. Additional HP-HIL Support**

| New Feature | New Keyword (Binary Required) | Tutorial Information |
|---|---|---|
| Capability of setting up interrupts for and communicating with many HP-HIL devices (useful when writing your own HP-HIL device drivers) | ON HIL EXT, HIL SEND, and HILBUF$ (KBD binary) | "HIL Devices" chapter of *BASIC Interfacing Techniques* |
| Capability of setting up interrupts for and reading pulses from the HP 46085A Control Dial (9-knob) Box | ON CDIAL, CDIAL, and OFF CDIAL (KBD binary) | "Communicating with the Operator" chapter in volume 1 of this manual |

## Additional Graphics Features

The following graphics features have been added to the BASIC system.

**Table 16-9. Additional Graphics Features**

| New Feature | Keyword (Binary Required) | Tutorial Information |
|---|---|---|
| New register that disables scrolling the display (to avoid scrolling graphics on bit-mapped alpha displays) | CRT register 16 | "Introduction to Graphics" chapter of *BASIC Graphics Techniques* |
| Can now send HPGL commands to PLOTTER IS device or file. | GSEND (GRAPH binary) | "Using Plotters and Printers" chapter of *BASIC Graphics Techniques* |
| Can simulate separate alpha and graphics rasters of Series 200/300 displays with a single statement (formerly required a short program) | SEPARATE ALPHA, MERGE ALPHA | "Using Graphics Effectively" chapter of *BASIC Graphics Techniques* |

## Additional CSUB Capabilities

The following capabilities have been added to CSUB's (Compiled Subroutines—created using the Pascal Workstation System and CSUB Utility).

**Table 16-10. Additional CSUB Capabilities**

| New Feature | General Capability | Tutorial Information |
|---|---|---|
| CSUB's can now perform I/O operations. | Categories of I/O procedures now available:<br>• Most of the Pascal I/O Procedure Library<br>• Some of the BASIC file I/O capabilities<br>• Some display I/O capabilities<br>• Some keyboard I/O capabilities<br>• All SYSTEM$ capabilities | *CSUB Utility* manual |

# Porting and Sharing Files 17

# Porting and Sharing Files 17

There are three different types of mass storage formats supported by BASIC:

- Logical Interchange Format (LIF)
- Shared Resource Manager (SRM)
- Hierarchical File System (HFS)

With each of these types of formats, BASIC supports three types of data files, as well as other types of files used by the BASIC system:

- ASCII
- BDAT
- HP-UX
- PROG
- BIN
- SYSTM

This chapter describes what tasks you will need to perform in transporting BASIC files from one type of volume to another. It also describes how to share HP-UX files between Series 200/300 BASIC, HP-UX, and Pascal systems.

## Sharing HFS Discs and Data Files

With the introduction of BASIC 5.0, it is now possible to share data files between BASIC applications and HP-UX applications using HFS volumes. This allows you to develop a total solution that takes advantage of the best features of each available operating system.

- As an example, a system can use BASIC for instrument control or automated data acquisition and then use HP-UX applications to analyze or manipulate the data for statistical quality control or management information systems.
- HP-UX also allows a gateway to networking capabilities that are becoming an important part of information sharing in the factory.
- Another advantage of HP-UX is the availability of the HP-UX Starbase Graphics Library and Graphics Hardware, which provides many additional graphics features that are not available with the BASIC Operating System.

### General Compatibility Requirements

In order to share data files between BASIC and HP-UX, there must be compatibility of:

- *File types* (both operating systems must be able to read and write a file to be shared)
- *Data representations* (both operating systems must write and interpret the bytes in the file in the same manner)

These requirements will be explored here, and examples of sharing data files between BASIC and HP-UX will be shown.

#### A Note About HP-UX File Terminology

From the following matrices, we see that BASIC and HP-UX can easily share files of "type HP-UX":

- On BASIC, these files will be listed with CAT as being of type `HP-UX`.
- On HP-UX, these files will be listed as `text` or `data`, depending on the *contents* of the file.

From the HP-UX viewpoint, this type of file can be called an "HP-UX text" file or an "HP-UX binary" file. The "HP-UX text" file contains data written in ASCII representation, while the HP-UX binary" file contains data written in internal representation.

## Common File Types

The following matrix shows which file types are supported by each operating system available on Series 300 computers.

**Table 17-1. Data File Support Matrix**

| Operating System or Language | ASCII | BDAT | Pascal Text[1] | HP-UX |
|---|---|---|---|---|
| BASIC 4.0 (or earlier) | Y | Y | | |
| BASIC 5.0 (or later) | Y | Y | | Y |
| Workstation Pascal 3.12 (or earlier) | Y | | Y | |
| Workstation Pascal 3.2 (or later) | Y | | Y | Y |
| Technical BASIC | | | | 2 |
| HP-UX C | 3 | | | Y |
| HP-UX Pascal | 3 | | | Y |
| HP-UX FORTRAN | 3 | | | Y |
| MS-DOS | 4 | | | |

**Legend:**

| | |
|---|---|
| 1 | "Pascal Text" files include type ".TEXT" files and type "Data" files that contain text. |
| Y | means that the Operating System or Language can easily read or write the file type with a native language program. |
| 2 | HP-UX Technical BASIC can only handle HP-UX files that contain text. |
| 3 | HP-UX has utilities to transfer LIF files to HFS volumes (*lifcp*, |
| 4 | Utilities are available for MS-DOS to transfer LIF files. |

## Common Data Types

Once a common file type to be used has been identified, the next step is to determine the data types that can be used within the file. To share data within a file between BASIC and HP-UX, the data type must be a type that is supported in both operating systems. The following matrix shows which data types are supported by each operating system available on Series 200/300 computers.

**Table 17-2. Data-Type Support Matrix**

| Operating System or Language | 16-Bit Integer | 32-Bit Integer | 32-Bit Real | 64-Bit Real | 128-Bit Complex[1] | String | Null-Terminated String |
|---|---|---|---|---|---|---|---|
| BASIC 4.0 (and earlier) | Y | | | Y | | Y | |
| BASIC 5.0 (and later) | Y | | | Y | Y | Y | [2] |
| Workstation Pascal 3.12 (and earlier) | Y | Y | | Y | | Y | |
| Workstation Pascal 3.2 (and later) | Y | Y | | Y | | Y | |
| Technical BASIC | | Y | Y | Y | | Y | |
| HP-UX C | Y | Y | Y | Y | | Y | Y |
| HP-UX Pascal | Y | Y | Y | Y | | Y | |
| HP-UX Fortran | Y | Y | Y | Y | Y | Y | Y |
| MS-DOS | Y | Y | Y | Y | | Y | Y |

**Legend:**

| | |
|---|---|
| 1 | The 128-bit complex data type is equivalent to two 64-bit reals. |
| Y | means that the Operating System or Language can easily read or write the data type with a native language program. |
| 2 | This data type works with "HP-UX binary" data files only. |

From this matrix, we see that BASIC and HP-UX can easily share data that is 16-bit integer, 64-bit real, 128-bit complex, string, and null-terminated strings. Before you can access this data, however, you must know:

- Which data types are used in the file.
- The order in which they are used.

Then you can use the corresponding data types in the programming language while reading the data. For example, BASIC and HP-UX C must have this data type matching to share data:

**Table 17-3. Data-Type Matching Between BASIC and C**

| BASIC | C |
|---|---|
| INTEGER | short |
| REAL | double |
| COMPLEX | 2 double's |
| String | array of char |

## HP-UX Text and Binary Files

"HP-UX text and binary" files are the native file types supported by HP-UX on HFS volumes. Support for this data file type has been added in BASIC 5.0 to allow sharing data files with HP-UX applications. BASIC still retains full support for all existing data file types, ASCII and BDAT, but some keywords have been updated to provide support for HFS discs and HP-UX text and binary files.

In particular, the ASSIGN, OUTPUT, and ENTER keywords now support "HP-UX text" and "HP-UX binary" files. (Note once again that these are both considered to be an HP-UX file *type* in BASIC; the only difference is in the file *contents*.)

A new CREATE statement has also been added to allow HP-UX files to be created from the BASIC system. To create an HP-UX file, use the CREATE keyword without the BDAT or ASCII secondary keywords.

```
CREATE "HPUX_file",10
```

When the ASSIGN statement is executed to open a file, the file type in the file header is examined. If the file is BDAT or ASCII, it will be treated as such. Otherwise, the file will be treated as:

- An “HP-UX binary” file (if it is assigned with FORMAT OFF)
- An “HP-UX text” file (if it is assigned with FORMAT ON).

## Examples of HP-UX File Access: Textual Numeric Data

Some examples will demonstrate how to access an HP-UX text file from BASIC and from HP-UX. The first program below is a BASIC program that writes some real numbers into an HP-UX text file.

```
10    !  RE-STORE "SHARE_TEXT"
20    !
30    !  Create & Assign the output file.
40    !
50    CREATE "TEXT_DATA",1                 ! Create an HP-UX file.
60    ASSIGN @File TO "TEXT_DATA";FORMAT ON  ! Treat as "text" file.
70    !
80    !  Output the data to the HP-UX Text file.
90    !
100   FOR N=-9.0 TO 8.5 STEP .07
110     OUTPUT @File;N
120   NEXT N
130   ASSIGN @File TO *
140   !
150   END
```

In this BASIC program, the file TEXT_DATA is an HP-UX file into which this program writes 250 real numbers. The ASSIGN statement is performed with FORMAT ON, thus specifying that this file is to be treated as a “text” file—using the ASCII data representation. (Note that the default FORMAT attribute for an HP-UX file is FORMAT OFF.) This program also demonstrates that file access of an “HP-UX text” file is performed with the same statements that would be used for access of an ASCII or BDAT file.

The next program is an HP-UX C program to read the data file that the above BASIC program wrote.

```
#include <stdio.h>
main()
{
        float   X, Y;
        FILE    *datafile, *fopen();

        /***                         ***/
        /*** Open file to read data ***/
        /***                         ***/
        datafile = fopen("/users/workstation/basic/files/TEXT_DATA", "r");
        if (datafile == NULL) {
                printf("Can't open file.\n");
                exit(1);
        }
        /***                                     ***/
        /*** Get data from file and print data ***/
        /***                                     ***/
        for (X = 1.0; X <= 250.0; X += 1.0) {
                fscanf(datafile, "%f", &Y);
                printf("%f\n",Y);
        }
        fclose(datafile);
}
```

In this HP-UX C example, the file TEXT_DATA is the "HP-UX text" file into which the BASIC program wrote 250 real numbers. Note that the HP-UX C program reads these real numbers as strings with the "fscanf" routine, then converts each string back to the real number value with the "%f" conversion specification.

**Note**

Data in an "HP-UX text" file is stored as ASCII characters, and this data can be read and edited by HP-UX editors or read by HP-UX commands such as "cat" and "more".

The next program is an HP-UX Technical BASIC program to read the data file which the preceding BASIC program wrote.

```
10 CLEAR
20 REAL x,y
30 name$="TEXT_DATA"
40 ASSIGN 14 TO name$
50 FOR x=1 TO 250
60 ENTER 14 ; y
70 PRINT x,y
80 NEXT x
90 ASSIGN 14 TO "*"
100 END
```

This HP-UX Technical BASIC example reads the file TEXT_DATA into which the BASIC program wrote real numbers. Note that Technical BASIC can convert each string back to the real number value with the number builder in the ENTER statement. This program demonstrates the simplicity of HP-UX Technical BASIC when used for sharing files between HP-UX Technical BASIC and the BASIC workstation environment.

Below is the hexadecimal dump of the first 40 bytes of the file TEXT_DATA, which will be used to get a better understanding of how BASIC formatted the data when it wrote to this "HP-UX text" file; contrast it to the following ASCII dump of the same file. (A listing of the program is shown at the end of this chapter.)

**HP-UX Text File Contents**

```
Contents of TEXT_DATA

BYTE|  +0  |  +1  |  +2  |  +3  |  +4  |  +5  |  +6  |  +7  |  +8  |  +9  |
----+------+------+------+------+------+------+------+------+------+------+
   0 |  2d  |  39  |   d  |   a  |  2d  |  38  |  2e  |  39  |  33  |   d  |
  10 |   a  |  2d  |  38  |  2e  |  38  |  36  |   d  |   a  |  2d  |  38  |
  20 |  2e  |  37  |  39  |   d  |   a  |  2d  |  38  |  2e  |  37  |  32  |
  30 |   d  |   a  |  2d  |  38  |  2e  |  36  |  35  |   d  |   a  |  2d  |
```

Here are the contents of the first 40 bytes of the file TEXT_DATA, shown in hexadecimal format. To show that the data items in an "HP-UX text" file are ASCII characters, the ASCII equivalent of this same data is now shown below.

**ASCII Character Equivalent**

| BYTE | +0 | +1 | +2 | +3 | +4 | +5 | +6 | +7 | +8 | +9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | - | 9 | "CR" | "LF" | - | 8 | . | 9 | 3 | "CR" |
| 10 | "LF" | - | 8 | . | 8 | 6 | "CR" | "LF" | - | 8 |
| 20 | . | 7 | 9 | "CR" | "LF" | - | 8 | . | 7 | 2 |
| 30 | "CR" | "LF" | - | 8 | . | 6 | 5 | "CR" | "LF" | - |

This table shows that a real number output to the HP-UX text file by BASIC is output as a string of ASCII characters representing the real number and this real number string terminated by "CR" and "LF" characters.

## Examples of HP-UX File Access: Textual Strings

The next data type to be demonstrated will be strings. The first program below is a BASIC program that writes some strings into an "HP-UX text" file.

```
10    !  RE-STORE "SHARE_STR2"
20    !
30    INTEGER N
40    !
50    !  Create & Assign the output file.
60    !
70    CREATE "STR2_DATA",1                  ! Create HP-UX file.
80    ASSIGN @File TO "STR2_DATA";FORMAT ON  ! Treat as "text" file.
90    !
100   !  Output the strings to the data file
110   !
120   FOR N=-9 TO 240
130     OUTPUT @File;"This is "&TRIM$(VAL$(N))&" line"
140   NEXT N
150   ASSIGN @File TO *
160   !
170   END
```

In this BASIC example, the file STR2_DATA is an "HP-UX text" file into which this program writes 250 data strings. The ASSIGN statement is again performed with FORMAT ON to specify that the data are to be represented in ASCII format (an "HP-UX text" file).

The next program is an HP-UX C program to read the data file that the above BASIC program wrote.

```
#include <stdio.h>
main()
{
        int     X;
        char    Strng[40];
        FILE    *datafile, *fopen();

        /***                       ***/
        /*** Open file to read data ***/
        /***                       ***/
        datafile = fopen("/users/workstation/basic/files/STR2_DATA", "r");
        if (datafile == NULL) {
                printf("Can't open file.\n");
                exit(1);
        }
        /***                                         ***/
        /*** Get string data from file and print data ***/
        /***                                         ***/
        for (X = 0; X <= 249; X += 1) {
                fgets(Strng, 40, datafile);
                printf("%s",Strng);
        }
        fclose(datafile);
}
```

In this HP-UX C example, the file STR2_DATA is the "HP-UX text" file into which the BASIC program wrote 250 data strings. Note that the HP-UX C program reads these data strings into an "array of char" with the "fgets" routine. The "fgets" routine used here terminates with the new-line character, then replaces this new-line character with a NULL character.

Below is the hexadecimal dump of the first 40 bytes of the file STR2_DATA, which will be used to get a better understanding of how BASIC stored the strings when it wrote to this "HP-UX text" file.

**HP-UX Text File Contents with Strings**

```
BYTE|  +0  |  +1  |  +2  |  +3  |  +4  |  +5  |  +6  |  +7  |  +8  |  +9  |
----+------+------+------+------+------+------+------+------+------+------+
   0|  54  |  68  |  69  |  73  |  20  |  69  |  73  |  20  |  2d  |  39  |
  10|  20  |  6c  |  69  |  6e  |  65  |   d  |   a  |  54  |  68  |  69  |
  20|  73  |  20  |  69  |  73  |  20  |  2d  |  38  |  20  |  6c  |  69  |
  30|  6e  |  65  |   d  |   a  |  54  |  68  |  69  |  73  |  20  |  69  |
```

These are the contents of the first 40 bytes of the file STR2_DATA, shown in hexadecimal format. To prove that the data in an "HP-UX text" file is ASCII characters, the ASCII equivalent of this same data is now shown below.

**ASCII Character Equivalent**

```
BYTE|  +0  |  +1  |  +2  |  +3  |  +4  |  +5  |  +6  |  +7  |  +8  |  +9  |
----+------+------+------+------+------+------+------+------+------+------+
   0|  T   |  h   |  i   |  s   |      |  i   |  s   |      |  -   |  9   |
  10|      |  l   |  i   |  n   |  e   | "CR" | "LF" |  T   |  h   |  i   |
  20|  s   |      |  i   |  s   |      |  -   |  8   |      |  l   |  i   |
  30|  n   |  e   | "CR" | "LF" |  T   |  h   |  i   |  s   |      |  i   |
```

This table shows that a string output to the HP-UX text file by BASIC is output as a string of ascii characters with no added length header bytes and terminated by "CR" (carriage-return) and "LF" (line-feed) characters.

## Examples of HP-UX File Access: Binary Real Values

There are three different types of data that can be stored in an HP-UX file that BASIC can also access. These data types are REAL, INTEGER, and string. The first type to be demonstrated in examples will be files of REAL data. The first program below is a BASIC program that writes some REAL numbers into an "HP-UX binary" file.

```
10    !  RE-STORE "SHARE_REAL"
20    !
30    !  Create & Assign the output file.
40    !
50    CREATE "REAL_DATA",1
60    ASSIGN @File TO "REAL_DATA";FORMAT OFF ! Treat as "binary" file.
70    !
80    !  Output the real numbers to the data file
90    !
100   FOR N=-9.0 TO 8.5 STEP .07
110     OUTPUT @File;N
120   NEXT N
130   ASSIGN @File TO *
140   !
150   END
```

In this BASIC program, the file REAL_DATA is an HP-UX file into which this program writes 250 real numbers. Note that the ASSIGN statement is performed with FORMAT OFF, thus specifying that this file is to be written as internal representation numbers. This program also demonstrates that file access of an "HP-UX binary" file is performed with the same keywords that would be used for access of a BDAT file.

The next program is an HP-UX C program to read the data file that the above BASIC program wrote.

```
#include <stdio.h>
main()
{
        int     X;
        double  Y[250];
        FILE    *datafile, *fopen();

        /***                          ***/
        /*** Open file to read data ***/
        /***                          ***/
        datafile = fopen("/users/workstation/basic/files/REAL_DATA", "r");
        if (datafile == NULL) {
                printf("Can't open file.\n");
                exit(1);
        }
        /***                                        ***/
        /*** Get real data from file and print data ***/
        /***                                        ***/
        fread((char *)Y, sizeof(Y[0]), 250, datafile);
        for (X = 0; X <= 249; X += 1)
                printf("%f\n",Y[X]);
        fclose(datafile);
}
```

In this HP-UX C example, the file REAL_DATA is an "HP-UX binary" file into which the BASIC program wrote 250 real numbers. Note that the HP-UX C program reads these real numbers into an array of double with the "fread" routine. This data must be handled as type double in C to remain compatible with the 64-bit real format used in BASIC. This data *cannot* be read and edited by HP-UX editors or read by HP-UX commands such as "cat" and "more". However, this data representation may allow for more efficient disc space use since every real number takes 8 bytes of disc space. The I/O transfer rates are also higher, since neither the output or the input routines need to format the data. In many cases, the internally represented numbers provide greater accuracy than would an ASCII representation of the number.

Below is the hexadecimal dump of the first 40 bytes of the file REAL_DATA, which will be used to get a better understanding of how BASIC represented the real number data when it wrote to this HP-UX (or "HP-UX binary") file.

**HP-UX Binary File Contents with Real Numbers**

```
BYTE|  +0  |  +1  |  +2  |  +3  |  +4  |  +5  |  +6  |  +7  |  +8  |  +9  |
----+------+------+------+------+------+------+------+------+------+------+
   0 |  c0  |  22  |   0  |   0  |   0  |   0  |   0  |   0  |  c0  |  21  |
  10 |  dc  |  28  |  f5  |  c2  |  8f  |  5c  |  c0  |  21  |  b8  |  51  |
  20 |  eb  |  85  |  1e  |  b8  |  c0  |  21  |  94  |  7a  |  e1  |  47  |
  30 |  ae  |  14  |  c0  |  21  |  70  |  a3  |  d7  |   a  |  3d  |  70  |
```

The real number data in an HP-UX binary file is formatted in IEEE-standard, 64-bit, floating-point notation for real numbers.

## Examples of HP-UX File Access: Binary Integers

The next data type to be demonstrated in examples will be files of integer data. The first program below is a BASIC program that writes some integer values into an HP-UX file.

```
10     !  RE-STORE "SHARE_INT"
20     !
30     INTEGER N
40     !
50     !  Create & Assign the output file.
60     !
70     CREATE "INT_DATA",1
80     ASSIGN @File TO "INT_DATA";FORMAT OFF ! Treat as "binary" file.
90     !
100    !  Output the integer numbers to the data file
110    !
120    FOR N=-9 TO 240
130      OUTPUT @File;N
140    NEXT N
150    ASSIGN @File TO *
160    !
170    END
```

In this BASIC program, the file INT_DATA is an HP-UX file into which this program writes 250 integer numbers. The ASSIGN statement is again performed with FORMAT OFF to specify that the internal data representation is to be used (which makes the file an "HP-UX binary" file).

The next program is an HP-UX C program that reads the data file that the above BASIC program wrote.

```
#include <stdio.h>
main()
{
        int     X;
        short   Y[250];
        FILE    *datafile, *fopen();

        /***                         ***/
        /*** Open file to read data ***/
        /***                         ***/
        datafile = fopen("/users/workstation/basic/files/INT_DATA", "r");
        if (datafile == NULL) {
                printf("Can't open file.\n");
                exit(1);
        }
        /***                                       ***/
        /*** Get integer data from file and print data ***/
        /***                                       ***/
        fread((char *)Y, sizeof(Y[0]), 250, datafile);
        for (X = 0; X <= 249; X += 1)
                printf("%d\n",Y[X]);
        fclose(datafile);
}
```

In this HP-UX C example, the file INT_DATA is the HP-UX binary (or untyped) file into which the BASIC program wrote 250 integer numbers. Note that the HP-UX C program reads these integers into an array of short with the "fread" routine. This data must be handled as type short in C to remain compatible with the 16-bit integer format used in BASIC. An HP-UX binary file allows more efficient disc-space use than an HP-UX text file, since each integer number takes 2 bytes of disc space. An HP-UX binary file is also faster because no format-conversion is required.

Below is the hex dump of the first 40 bytes of the file "INT_DATA", which will be used to get a better understanding of how BASIC formatted the integer number data when it wrote to this HP-UX binary file.

**HP-UX Binary File Contents with INTEGER Values**

```
BYTE|  +0  |  +1  |  +2  |  +3  |  +4  |  +5  |  +6  |  +7  |  +8  |  +9  |
----+------+------+------+------+------+------+------+------+------+------+
   0|  ff  |  f7  |  ff  |  f8  |  ff  |  f9  |  ff  |  fa  |  ff  |  fb  |
  10|  ff  |  fc  |  ff  |  fd  |  ff  |  fe  |  ff  |  ff  |   0  |   0  |
  20|   0  |   1  |   0  |   2  |   0  |   3  |   0  |   4  |   0  |   5  |
  30|   0  |   6  |   0  |   7  |   0  |   8  |   0  |   9  |   0  |   a  |
```

INTEGERs in an HP-UX binary file are formatted in 16-bit two's-complement notation.

## Examples of HP-UX File Access: Binary Strings

The first program below is a BASIC program that writes some strings into an HP-UX binary file.

```
10     !  RE-STORE "SHARE_STR"
20     !
30     INTEGER N
40     !
50     !  Create & Assign the output file.
60     !
70     CREATE "STR_DATA",1
80     ASSIGN @File TO "STR_DATA";FORMAT OFF ! Treat as "binary" file.
90     !
100    !  Output the strings to the data file
110    !
120    FOR N=-9 TO 240
130      OUTPUT @File;"This is "&TRIM$(VAL$(N))&" line"
140    NEXT N
150    ASSIGN @File TO *
160    !
170    END
```

In this BASIC example, the file STR_DATA is an HP-UX file into which this program writes 250 data strings. The ASSIGN statement is again performed with FORMAT OFF to specify that the internal data representations are to be used (an "HP-UX binary" file). Each string output to the file has a null character, CHR$(0), appended to the end of the string automatically by the OUTPUT statement. This null character is used by the HP-UX C program as a string-termination character.

The next program is an HP-UX C program to read the data file that the above BASIC program wrote.

```
#include <stdio.h>
main()
{
        int     I, X;
        char    Strng[40];
        FILE    *datafile, *fopen();

        /***                        ***/
        /*** Open file to read data ***/
        /***                        ***/
        datafile = fopen("/users/workstation/basic/files/STR_DATA", "r");
        if (datafile == NULL) {
                printf("Can't open file.\n");
                exit(1);
        }
        /***                                         ***/
        /*** Get string data from file and print data ***/
        /***                                         ***/
        for (X = 0; X <= 249; X += 1) {
                I = 0;
                while ((Strng[I] = getc(datafile)) != '\000')
                        I++;
                printf("%s\n",Strng);
        }
        fclose(datafile);
}
```

In this HP-UX C example, the file STR_DATA is the HP-UX file into which the BASIC program wrote 250 strings. Note that the HP-UX C program reads these strings into an "array of char" with the "getc" routine reading each character. The "while" loop repeats until a null character has been read by the "getc" routine.

Below is the hexadecimal dump of the first 60 bytes of the file "STR_DATA". This shows how BASIC formatted the strings in this HP-UX binary file.

**HP-UX Binary File Contents with Strings**

```
BYTE|  +0  |  +1  |  +2  |  +3  |  +4  |  +5  |  +6  |  +7  |  +8  |  +9  |
----+------+------+------+------+------+------+------+------+------+------+
  0 |  54  |  68  |  69  |  73  |  20  |  69  |  73  |  20  |  2d  |  39  |
 10 |  20  |  6c  |  69  |  6e  |  65  |   0  |  54  |  68  |  69  |  73  |
 20 |  20  |  69  |  73  |  20  |  2d  |  38  |  20  |  6c  |  69  |  6e  |
 30 |  65  |   0  |  54  |  68  |  69  |  73  |  20  |  69  |  73  |  20  |
 40 |  2d  |  37  |  20  |  6c  |  69  |  6e  |  65  |   0  |  54  |  68  |
 50 |  69  |  73  |  20  |  69  |  73  |  20  |  2d  |  36  |  20  |  6c  |
```

To help visualize how this data is stored in an HP-UX binary file, the ASCII equivalent of this same data is now shown below.

**ASCII Character Equivalent**

| BYTE | +0 | +1 | +2 | +3 | +4 | +5 | +6 | +7 | +8 | +9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | T | h | i | s |  | i | s |  | - | 9 |
| 10 |  | l | i | n | e | "NUL" | T | h | i | s |
| 20 |  | i | s |  | - | 8 |  | l | i | n |
| 30 | e | "NUL" | T | h | i | s |  | i | s |  |
| 40 | - | 7 |  | l | i | n | e | "NUL" | T | h |
| 50 | i | s |  | i | s |  | - | 6 |  | l |

This table shows that each string written by BASIC was terminated by a null character. There is no carriage return or line feed.

## Examples of ASCII File Access

The file types demonstrated so far have been HP-UX files that both BASIC and HP-UX can easily access with a native language program. When the file type is a ASCII file, it can still be accessed from HP-UX.

This example writes real numbers into a ASCII file.

```
10    ! RE-STORE "SHARE_ASC"
20    !
30    ! Create & Assign the output file.
40    !
50    CREATE ASCII "ASC_DATA",1
60    ASSIGN @File TO "ASC_DATA"
70    !
80    ! Output the data to the ASCII file.
90    !
100   FOR N=-9.0 TO 8.5 STEP .07
110     OUTPUT @File;N
120   NEXT N
130   ASSIGN @File TO *
140   !
150   END
```

The next program is an HP-UX C program to read the data file that the above BASIC program wrote.

```
#include <stdio.h>
#include <math.h>
main()
{
        float   Y, Result, rval;
        short   I[1];
        int     J, X;
        char    Strng[40];
        FILE    *datafile, *fopen();

        /***                         ***/
        /*** Open file to read data ***/
        /***                         ***/
        datafile = fopen("/users/workstation/basic/files/ASC_DATA", "r");
        if (datafile == NULL) {
                printf("Can't open file.\n");
                exit(1);
        }
        /***                                          ***/
        /*** Get voltage data from file and print data ***/
        /***                                          ***/
        fseek(datafile, 512, 0);
        for (X = 1; X <= 250; X += 1) {
                fread((char *)I, sizeof(I), 1, datafile);
                rval = I[0];
                if ((Result = fmod(rval, 2.0)) != 0.0)
                        I[0] ++;
                J = 0;
                while (J < I[0]) {
                        Strng[J] = getc(datafile);
                        J++;
                }
                sscanf(Strng, "%f", &Y);
                printf("%f\n",Y);
        }
        fclose(datafile);
}
```

In this HP-UX C example, the file ASC_DATA is the ASCII file into which the BASIC program wrote 250 real values. Note that the HP-UX C program reads the 2-byte length header with the "fread" routine, then uses this length number to read the same number of characters with the "getc" routine. The "sscanf" routine then converts each string back to the real number value with the "%f" conversion specification. This program also requires the "fseek" routine to force the file pointer to skip over the 512-byte header block that BASIC inserts at the beginning of the ASCII file.

Below is the hexadecimal dump of significant portions of the first 560 bytes of the file ASC_DATA, which shows how BASIC formatted the data when it wrote to this ASCII file.

**ASCII File Contents with Real Values**

```
BYTE|  +0  |  +1  |  +2  |  +3  |  +4  |  +5  |  +6  |  +7  |  +8  |  +9  |
----+------+------+------+------+------+------+------+------+------+------+
   0|  80  |   0  |  48  |  46  |  53  |  4c  |  49  |  46  |   0  |   0  |
  10|   0  |   1  |  10  |   0  |   0  |   0  |   0  |   0  |   0  |   1  |
  20|   0  |   0  |   0  |   0  |   0  |   0  |   0  |   1  |   0  |   0  |
  30|   0  |   1  |   0  |   0  |   0  |   3  |  11  |  11  |  11  |  11  |
  40|  11  |  11  |   0  |   0  |   0  |   0  |   0  |   0  |   0  |   0  |
  .
  .
  .
 240|   0  |   0  |   0  |   0  |   0  |   0  |   0  |   0  |  11  |  11  |
 250|  11  |  11  |  11  |  11  |   0  |   0  |  57  |  53  |  5f  |  46  |
 260|  49  |  4c  |  45  |  20  |  20  |  20  |   0  |   1  |   0  |   0  |
 270|   0  |   2  |   0  |   0  |   0  |   3  |  86  |  12  |   5  |  15  |
 280|  54  |  16  |  80  |   1  |   0  |   0  |   0  |   0  |   0  |   0  |
 290|   0  |   0  |   0  |   0  |   0  |   0  |   0  |   0  |  ff  |  ff  |
  .
  .
  .
 510|   0  |   0  |   0  |   2  |  2d  |  39  |   0  |   5  |  2d  |  38  |
 520|  2e  |  39  |  33  |  20  |   0  |   5  |  2d  |  38  |  2e  |  38  |
 530|  36  |  20  |   0  |   5  |  2d  |  38  |  2e  |  37  |  39  |  20  |
 540|   0  |   5  |  2d  |  38  |  2e  |  37  |  32  |  20  |   0  |   5  |
 550|  2d  |  38  |  2e  |  36  |  35  |  20  |   0  |   5  |  2d  |  38  |
```

The ASCII equivalent of this same data is shown below.

**ASCII Character Equivalent**

| BYTE | +0 | +1 | +2 | +3 | +4 | +5 | +6 | +7 | +8 | +9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 80hex | "NUL" | H | F | S | L | I | F | "NUL" | "NUL" |
| 10 | "NUL" | "SOH" | "DLE" | "NUL" | "NUL" | "NUL" | "NUL" | "NUL" | "NUL" | "SOH" |
| 20 | "NUL" | "NUL" | "NUL" | "NUL" | "NUL" | "NUL" | "NUL" | "SOH" | "NUL" | "NUL" |
| 30 | "NUL" | "SOL" | "NUL" | "NUL" | "NUL" | "EXT" | "DC1" | "DC1" | "DC1" | "DC1" |
| 40 | "DC1" | "DC1" | "NUL" | "NUL" | "NUL" | "NUL" | "NUL" | "NUL" | "NUL" | "NUL" |
| . | | | | | | | | | | |
| . | | | | | | | | | | |
| . | | | | | | | | | | |
| 240 | "NUL" | "NUL" | "NUL" | "NUL" | "NUL" | "NUL" | "NUL" | "NUL" | "DC1" | "DC1" |
| 250 | "DC1" | "DC1" | "DC1" | "DC1" | "NUL" | "NUL" | W | S | _ | F |
| 260 | I | L | E | | | | "NUL" | "SOH" | "NUL" | "NUL" |
| 270 | "NUL" | "STX" | "NUL" | "NUL" | "NUL" | "EXT" | 86hex | "DC2" | "ENQ" | "NAK" |
| 280 | T | "SYNC" | 80hex | "SOH" | "NUL" | "NUL" | "NUL" | "NUL" | "NUL" | "NUL" |
| 290 | "NUL" | "NUL" | "NUL" | "NUL" | "NUL" | "NUL" | "NUL" | "NUL" | FFhex | FFhex |
| . | | | | | | | | | | |
| . | | | | | | | | | | |
| . | | | | | | | | | | |
| 510 | "NUL" | "NUL" | "NUL" | "STX" | - | 9 | "NUL" | "ENQ" | - | 8 |
| 520 | . | 9 | 3 | | "NUL" | "ENQ" | - | 8 | . | 8 |
| 530 | 6 | | "NUL" | "ENQ" | - | 8 | . | 7 | 9 | |
| 540 | "NUL" | "ENQ" | - | 8 | . | 7 | 2 | | "NUL" | "ENQ" |
| 550 | - | 8 | . | 6 | 5 | | "NUL" | "ENQ" | - | 8 |

This table shows the 512 byte header block that BASIC puts at the beginning of a LIF ASCII file on HFS discs. The portions of this block not shown in the table contains all zeros. This table also shows the 2-byte length header at the beginning of each string. Note that the strings have no added termination characters.

### HP-UX File Dump Utility

When debugging problems that can arise from BASIC and HP-UX sharing data files, it may be necessary to look at the contents of data within the file. This program is an HP-UX C program to read the contents of a file and display the contents in hexadecimal format.

```
#include <stdio.h>
main(argc, argv)
int     argc;
char    *argv[];
{
        int     X, Y, values[20], start, end;
        FILE    *datafile, *fopen();

        printf("Contents of %s\n",argv[1]);
        /***                        ***/
        /*** Open file to read data ***/
        /***                        ***/
        datafile = fopen(argv[1], "r");
        if (datafile == NULL) {
                printf("Can't open file %s\n",argv[1]);
                exit(1);
        }
        /***                                    ***/
        /*** Get data from file and print data ***/
        /***                                    ***/
        sscanf(argv[2], "%d", &start);
        sscanf(argv[3], "%d", &end);
        fseek(datafile, start, 0);
        printf("BYTE|");
        for (X = 0; X <= 9; X += 1) {
                printf("%4+d  |",X);
        }
        printf("\n");
        for (X = 0; X <= 3; X += 1) {
                printf("-");
        }
        printf("+");
        for (Y = 0; Y <= 9; Y += 1) {
                for (X = 0; X <= 5; X += 1) {
                        printf("-");
                }
                printf("+");
        }
        printf("\n");
        for (Y = start; Y <= end; Y +=10) {
                printf("%3d |",Y);
                for (X = 0; X <= 9; X += 1) {
                        values[X] = getc(datafile);
                        values[X] = values[X] & 0377;
                        printf("%4x  |",values[X]);
                }
                printf("\n");
        }
        fclose(datafile);
}
```

Once this program has been compiled, it is executed by the following syntax:

*prog_name file_name start end*

in which:

| | |
|---|---|
| *prog_name* | is the name of the compiled program that is to be executed |
| *file_name* | is the file to dump |
| *start* | is the starting byte at which to begin the dump |
| *end* | is the last byte of the dump |

## Porting LIF Files to SRM

This section summarizes ways you can modify existing programs that use LIF discs to allow those programs to access the SRM system.

When modifying programs to access SRM-controlled mass storage device(s), you should be aware that:

- LIF and SRM mass storage file specifiers may differ and string variable names that contain file specifiers may need corresponding modification.
- References to mass storage volume specifiers (msvs) throughout the program may have to be altered.
- Allowances may have to be made for directory path specification.
- LIF protect codes may differ from SRM passwords. The syntax for protecting SRM files is different from that used for LIF files.

### SRM File Specifiers

#### Composition of SRM File Names

All file names for local mass storage are one to 10 characters long, while SRM file names contain one to 16 characters. Remote file names can contain the period character ( . ) while local files cannot. If file name compatibility between resources is required, use 10 or fewer characters and do not use periods within SRM file names.

#### SRM File and Mass Storage Device Specification in String Variables

Modifying programs for use with SRM resources generally requires changing the value, and often the length, of the string variables used to specify files and mass storage devices. The statements that assign the actual values to the string variables may have to be modified individually.

Some programs use one string variable for the entire file specifier. For instance:

```
100   DIM File_specifier$[32]
110   LINPUT "Enter file specifier", File_specifier$
120   ON ERROR GOTO 110 ! Try again if improper specifier.
130   ASSIGN @Path TO File_specifier$
140   OFF ERROR
```

If one variable is used for all file specifiers (as in the preceding example), only the length of the variable needs to be changed to allow for the additional characters allowed in SRM file specifiers.

The maximum number of characters that can be entered into a string variable from the keyboard in one operation is a good size for a file specifier variable.

- Series 200 Models 216, 220, 226 and 236 allow up to 160 characters. Series 300 computers with medium-resolution displays also allow 160 characters.
- Model 237 allows 256 characters. Series 300 computers with high-resolution displays also allow 256 characters.

Thus, the length of `File_specifier$` in the preceding example's DIM statement would be changed from 32 to 160 or 256, accordingly.

Note that the system mass storage device (the current MASS STORAGE IS device) will be accessed if no msvs is explicitly specified.

## SRM Mass Storage Volume Specification

Some programs use separate variables for the file name and volume specifiers. For example:

```
ASSIGN @Path TO Filename$&Msvs$
```

If so, both variables may have to be dimensioned to greater lengths. Allowing 34 characters for the file name variable accommodates a 16-character file name, a 16-character password, and the "<" and ">" password delimiters (for example, "ASCDEFGHIJ123456<1234567890123456>"). The SRM volume specifier may occupy up to 54 characters.

Other programs may use MASS STORAGE IS statements throughout the program instead of including the msvs in each file specifier. For instance:

```
MASS STORAGE IS Left_drive$
ASSIGN @File TO File_name$
```

Unless variable(s) are used to specify the msvs and each variable is assigned a value in only one place, you may have to modify each MASS STORAGE IS statement to specify the desired SRM volume.

## Allowing for SRM Directory Paths

Suppose the following program needs to be modified to include a SRM file's directory path.

```
100  DIM Filename$[14],Msvs$[20]
       .
       .
       .
500  Filename$="SLIDES"
510  Msvs$=":HP9895,700"
       .
       .
       .
1000 ASSIGN @File TO Filename$&Msvs$
1010 OUTPUT @File;Data(*)
1020 ASSIGN @File TO *
       .
       .
       .
2000 ASSIGN @File TO Filename$&Msvs$
2010 OUTPUT @File;Data(*)
2020 ASSIGN @File TO *
```

In the next example, it is probably easiest to add another string variable for the (optional) directory path name. For example:

```
100  DIM Dir_path$[160],File_name$[80],Vol_spec$[80]
       .
       .
       .
500  Dir_path$="FRED/DATA_FILES/"
510  File_name$="SLIDES"
520  Vol_spec$=":REMOTE 21,1"
       .
       .
       .
1000 ASSIGN @File TO Dir_path$&File_name$&Vol_spec$
1010 OUTPUT @File;Data(*)
1020 ASSIGN @File TO *
```

If the `Dir_path$` variable is null, the statement looks exactly like it did before the modification. If the `Vol_spec$` variable is null, the current mass storage device is accessed. The only difference is in the allowable length of the string variables.

## SRM Passwords vs. LIF Protect Codes

The PROTECT statement syntax for SRM files is different from the syntax for LIF files. Depending on the type of mass storage that is being used, you can use either of the following to decide which syntax will be used:

1. Try the non-SRM syntax with an ON ERROR statement enabled. If an error occurs, see if it indicates that the mass storage device is an SRM. An Error 1 occurs when the following statement is executed on an SRM file:

   `PROTECT` *file specifier,protect code*

2. If the program uses a string to store the volume specifier, check for a non-zero value of `POS(Vol_spec$,"REMOTE")`. This alternative is easier to implement than alternative 1 but will not work if the program accesses the default device when `Vol_spec$` is empty.

If the program looks for a password error (Error 62) at ASSIGN time, the program may have to be modified because the system may not detect the password error until an ENTER @Path or OUTPUT @Path is attempted.

## Copying Item-by-Item Using ENTER and OUTPUT

You may copy a file from LIF to SRM mass storage one item at a time, as illustrated in the programs that follow. These programs use the ENTER and OUTPUT statements to copy data item-by-item from a LIF BDAT file to an SRM BDAT file.

The first program creates and fills a BDAT file named `BDAT_FILE`.

```
10     CREATE BDAT "BDAT_FILE:INTERNAL",10
20     ASSIGN @Local TO "BDAT_FILE:INTERNAL"
30     !
40     FOR Item=1 TO 50
50     OUTPUT @Local;"String data item"
60     NEXT Item
70     !
80     ASSIGN @Local TO *
90     END
```

The second program copies the contents of `BDAT_FILE` item-by-item into a file (also called `BDAT_FILE`) in the SRM directory named `General` (shown in the previous illustration).

```
100    DIM String_item$[20]
110    CREATE BDAT "PROJECTS/General/BDAT_FILE:REMOTE",10
120    ASSIGN @Local TO "BDAT_FILE:INTERNAL"
130    ASSIGN @Remote TO "PROJECTS/General/BDAT_FILE:REMOTE"
140    !
150    FOR Item=1 TO 50
160    ENTER @ Local;String_item$
170    OUTPUT @Remote;String_item$
180    NEXT Item
190    !
200    ASSIGN @Local TO *
210    ASSIGN @Remote TO *
220    END
```

## Accessing Files Created on Non-Series-200/300 SRM Workstations

Regardless of the kind of the computer or language system, ASCII files can be shared among all workstations on the SRM.

This example shows how you can access an ASCII file named `Prog_x`, which was created on an HP 9845 with the SAVE ASCII statement.

In this example, `Prog_x` is in an HP 9845 workstation user's directory called `COMMON`. `COMMON` is located in the directory `WORK_45`, which is at the root of the SRM directory structure. The password `mypass` protects the READ capability on `WORK_45`. All access capabilities on `COMMON` are public.

To access `Prog_x` on a Series 200/300 Workstation, you would type:

```
GET "WORK_45<mypass>/COMMON/Prog_x:REMOTE"
  or
GET "/WORK_45<mypass>/COMMON/Prog_x"
```

The system would then put `Prog_x` into your workstation. Keep in mind that, with GET, any lines containing syntax that is invalid for Series 200/300 BASIC will be stored as commented program lines (such as `100 !  BEEPER 10,10` ).

# 5.1 Enhancements 18

# 5.1 Enhancements 18

BASIC 5.1 consists of functionality additions and manual enhancements.

## Functionality Additions

| New Feature | Description |
|---|---|
| PaintJet(TM) Support (HP 3630A Color Graphics Printer) | A CSUB performs a color dump. See the "BASIC Utilities Library" chapter of the *Installing and Maintaining the BASIC System* manual. |
| HP 98548A, HP 98549A, and HP 98550A Display Support | These cards are high resolution bit-mapped display interfaces. (Note that on these displays, the alpha cursor will not blink.) |
| New CSUB Utility Features | Passing COMPLEX and I/O-path-name parameters to CSUB's. See the *BASIC 5.1 CSUB Utility* manual. |
| HP 98646A VME Interface CSUB | This CSUB was formerly a separate product, but is now included in the BASIC 5.1 product (there are no new features). See the "BASIC Utilities Library" chapter of the *Installing and Maintaining the BASIC System* manual. |

## Manual Changes

In order to make the installation and maintenance of the BASIC Language System easier, the *Installing, Using and Maintaining the BASIC System* manual has been divided into two manuals:

- *Installing and Maintaining*
- *Using BASIC*

## Duplicating Files with the LINK Command

The COPY command creates a duplicate file by copying a file's contents under a different file name. On the other hand, the LINK command creates additional filenames which refer to the same file. This is done by creating:

- A new file name (an entry in a directory).
- A *pointer* to the existing file's contents (note that the contents are *not* duplicated).

Thus, LINK saves disc space, because there is only one copy of the body of the file (but there are two file names, or links, to the file's contents).

It is important to remember that the LINK command is usable only with HFS and SRM volumes.

### A Simple Example

Executing the following statement:

```
LINK "Existing_file" TO "New_name"
```

links the file name called `New_name` to the existing file called `Existing_file`.

### Characteristics of Linked Files

As mentioned before, LINK makes a new name and creates a pointer to an existing file. This gives the file some useful, but sometimes subtle, characteristics. For instance, an OUTPUT to a data file will change its contents. Using ENTER with any file linked to the changed file will reflect the change. (Note that there is only one copy of the file's data but two or more file names are linked to it.)

If you RE-STORE or RE-SAVE a file that is linked to other files, a new file will be created and the link to the original file will be *broken*. In this case the RE-STORE'd or RE-SAVE'd file is changed but the linked files are *not* changed.

The keyword LINK is like the COPY command and in some cases they can be used interchangeably. However, COPY results in two different files, while LINK results in one file with two different file names linked to it. LINK is used when you need to save disk space or you want the OUTPUT command to change the contents of all linked files. COPY should be used if you want OUTPUT to modify one version of the file but not all versions.

## More Examples of Creating Linked Files

The first example below creates an HP-UX file and links it to another file. It then outputs data to the original file. Finally the contents of each file (which are identical) are displayed.

```
100   CREATE "File_1",1          ! Create an HP-UX file.
110   !
120   LINK "File_1" TO "File_2"  ! Link "File_2" to "File_1".
130   !
140   ASSIGN @File_1 TO "File_1" ! Open File_1 for writing.
150   !
160   String$="This is a test."  ! Assign the output string.
170   OUTPUT @File_1;String$     ! Output message to all linked files.
180   !
190   ASSIGN @File_1 TO *        ! Close File_1 before reading it.
200   !
210   ASSIGN @File_1 TO "File_1"       ! Open File_1 for reading.
220   ENTER @File_1;String1$           ! Read @File_1 into String1$
230   PRINT "The contents in File_1: ";String1$ ! Print String1$
240   ASSIGN @File_1 TO *              ! Close File_1
250   !
260   ASSIGN @File_2 TO "File_2"       ! Open File_2 for reading.
270   ENTER @File_2;String2$           ! Read @File_2 into String2$
280   PRINT "The contents in File_2: ";String2$ ! Print String2$
290   ASSIGN @File_2 TO *              ! Close File_2
300   !
310   END
```

Here are the results of running the program:

```
The contents in File_1:  This is a test.

The contents in File_2:  This is a test.
```

**Example of Breaking a Link**

The next example shows how RE-STORE breaks a link. Create the program:

```
10 PRINT "Original Program"
20 END
```

Next give this file the name `Original` and store and create a link to it:

```
STORE "Original"
LINK "Original" TO "New_name"
```

Modify the program called `New_name` to read:

```
10 PRINT "Modified Program"
20 END
```

Now store the program using the RE-STORE command:

```
RE-STORE "New_name"
```

If you next LOAD the program called `Original` and LIST it, you will find that it remains unchanged because the link between `Original` and `New_name` is broken. (That is, there are now two files on the system, each being slightly different from the other.)

# BASIC/UX Differences and Enhancements 19

# BASIC/UX Differences and Enhancements 19

## Introduction

The HP Series 200/300 BASIC language system currently has three separate implementations:

| | |
|---|---|
| BASIC/WS | The Workstation implementation, which is a combined language and operating system that runs on HP 9000 Series 200/300 computers. (This is probably the most familiar implementation of HP Series 200/300 BASIC language.) |
| BASIC/DOS | The DOS implementation, which is a card that plugs into an HP Vectra or other PC-compatible computer. This implementation is essentially a Series 200 computer on a card that plugs into a PC. (The card contains the BASIC language/operating system in ROM.) |
| BASIC/UX | The HP-UX implementation, which is essentially the BASIC interpreter and part of the BASIC Workstation operating system that runs as a set of processes "on top of" the HP-UX operating system. |

This chapter describes the differences and enhancements that BASIC/UX provides to the BASIC/WS implementation. (It does *not* describe the differences between BASIC/DOS and BASIC/WS. See the documentation supplied with the HP 82321 BASIC Language Processor card for that information.)

## Prerequisites to Reading this Chapter

In order to understand the information in this chapter, you should:

- Already be familiar with the BASIC/WS implementation of HP BASIC (otherwise, this "delta" information will not make much sense to you).
- Have a copy of the BASIC/UX manual set available (so you can look up some of the details of the brief descriptions given here).

## Compatibility Between BASIC/UX and BASIC/WS

The BASIC/UX implementation is highly compatible with the BASIC/WS implementation. Nearly all programs written for BASIC/WS will run (with *minor* modifications) on the BASIC/UX system. However, because BASIC/WS has its own operating system and BASIC/UX runs on the HP-UX operating system:

- Some BASIC/WS keywords do not work on BASIC/UX.
- Some BASIC/WS keywords work differently on BASIC/UX.
- BASIC/UX has also added new keywords to the BASIC/WS keyword set.

This chapter provides a *brief summary* of these differences and enhancements. For a *detailed* description of a *particular* difference or enhancement, refer to the BASIC manual specified in the table entry (in this chapter) that describes that feature.

### Are You Porting BASIC/WS Programs to BASIC/UX ?

One of the main benefits of reading this chapter is that it will help you determine what will be required to port existing BASIC/WS programs to run on the BASIC/UX system. If you are porting existing software, you should at least scan this chapter *before* reading the chapter called "Porting BASIC/WS Programs to BASIC/UX".

## Summary of BASIC/UX Differences

The following tables list the differences between BASIC/UX and BASIC/WS keywords.

### Legend for Subsequent Tables

After each brief keyword description, the table gives a manual name and chapter number to help you locate more information on that particular keyword. Here are the acronyms used in the tables to specify BASIC manuals.

| | |
|---|---|
| GT | *BASIC Graphics Techniques* |
| IM | *Installing and Maintaining the BASIC/UX System* |
| IT | *BASIC/UX Interfacing Techniques* |
| LR | *BASIC Language Reference* |
| PT | *BASIC Programming Techniques* |
| UB | *Using the BASIC/UX System* |

| Keyword/Feature | BASIC/WS Description | BASIC/UX Description | For More Info |
|---|---|---|---|
| ALPHA ON and ALPHA OFF | Turns the alphanumeric display on or off. | On a console, these keywords function the same as for BASIC/WS. However, on a terminal or in X Windows these keywords are no-ops. The reason for this is on terminals alpha and graphics are separated and cannot be merged and in X Windows alpha and graphics are merged and cannot be separated. | LR Keyword Dictionary |
| ASSIGN | Assigns an I/O path name and attributes to a device, group of devices, a mass storage file, or a buffer. | Extended to allow piping to/from HP-UX commands (such as `"\| lp"`) and to allow output to windows. | IT Chapter 3 |
| CAT | Lists the contents of a mass storage directory or provides information on a specified PROG file. | For HFS directories, the device selector, unit, and volume numbers normally found in the first line are not displayed, and the device type is shown as HFS. Also, CAT on BASIC/UX recognizes additional file types which may be found on HFS volumes (e.g., PIPE). CAT has not changed for LIF or SRM directories. | LR Keyword Dictionary |
| CONTROL CRT,18 | Sets the alpha write-enable mask to a bit pattern. | Same as the description for the keywords ALPHA ON and ALPHA OFF. | IT Chapter 10 |
| CONTROL CRT,21 | Switches between Series 200 CRT compatibility mode and native bit-mapped mode. | Not supported. | IT Chapter 10 |

| Keyword/Feature | BASIC/WS Description | BASIC/UX Description | For More Info |
|---|---|---|---|
| CONTROL KBD,11 | Sets the knob pulse mode. | KNB2_0 binary functionality is not supported on BASIC/UX. | IT Chapter 11 |
| CONTROL 32 | This *pseudo* select code is used to turn on and off parity checking, system (memory) cache, and processor (instruction) cache. | Control registers 0, 1, and 3 are not supported on pseudo select code 32; however, all of its status registers are supported. | PT Chapter 13 |
| CREATE | Creates a new file. | The default permission for files and directories created on HFS volumes may be modified using the HP-UX **umask** (1) command. | HP-UX Language Reference |
| Datacomm and Serial Interfaces: • CONTROL Registers • STATUS Registers | Supported as stated in the *BASIC Interfacing Techniques* manual. | Some registers are not supported or have slightly different definitions, some registers have been added, and READIO and WRITEIO are not supported. Also, the HP 98642A 4-Channel Multiplexer has been added as a new datacomm interface card that is supported by BASIC/UX. | IT Chapter 13 and Chapter 14 |
| DUMP ALPHA DUMP GRAPHICS | Copies the output of the alpha or graphics display to a printing device. | Extended to support output to windows and named or unnamed pipes. Note that windows can only receive alpha output. Also, DUMP GRAPHICS is not supported on terminals. | GT Chapter 3 |
| DUMP DEVICE IS | Specifies which device receives the data when either DUMP ALPHA or DUMP GRAPHICS is executed without a device selector. | Extended to support output to windows and named or unnamed pipes. Note that windows can only receive alpha output. | GT Chapter 3 |

| Keyword/Feature | BASIC/WS Description | BASIC/UX Description | For More Info |
|---|---|---|---|
| ENTER | Used to input data from a specified source and assign the value(s) entered to variable(s). | Extended to support input from a named or unnamed pipe. | IT Chapter 5 |
| ERRM$ ERRN | ERRM$ returns the text of the error message associated with the most recent program execution error. ERRN returns the number of the most recent error. | BASIC/UX provides new errors in the range 186 to 198 and 810 to 867. (All error messages are listed in the appendices of the *BASIC Language Reference* and *BASIC Condensed Reference*.) | PT Chapter 11 |
| GESCAPE CRT, 7 | Sets the graphics write- and display-enable masks. | Not supported in windows or terminals. | GT Chapter 2 |
| GLOAD | Loads the contents of an INTEGER array into a frame buffer. | Not supported on terminals. | GT Chapter 2 |
| GRAPHICS INPUT IS | Assigns the graphics input device for DIGITIZE, READ LOCATOR, etc. | Works the same as with BASIC/WS on a console. In a window environment, use the keyword SET HIL MASK to select graphics input devices not used by the window manager. Terminals are restricted to input from the keyboard. | LR Keyword Dictionary |
| GRAPHICS ON GRAPHICS OFF | Turns the graphics display on or off. | Same as the description for the keywords ALPHA ON and ALPHA OFF. | LR Keyword Dictionary |
| GSTORE | Stores the contents of the frame buffer into an INTEGER array. | Not supported on terminals. | GT Chapter 2 |

| Keyword/Feature | BASIC/WS Description | BASIC/UX Description | For More Info |
|---|---|---|---|
| HFS Formatted Discs | Requires that HFS discs be connected to the system to access them. | Requires that all HFS discs be "mounted" before they can be accessed. This allows file buffering to be performed, which greatly improves performance. Also note that with BASIC/UX, all HFS volumes are joined into a single hierarchy (by the HP-UX operating system). In addition, STATUS register 2 is not supported for an I/O path name assigned to a file on an HFS volume. | UB Chapter 5 |
| HP-HIL Commands:<br>IDD<br>RNM<br>EXD<br>RST<br>RSC | These HP-HIL commands work as stated in Appendix A of the BASIC Workstation *Interfacing Techniques* manual. | Since BASIC/UX cannot determine the number of bytes in a packet of information returned when you execute these commands, it uses the null byte as a packet terminator. This means that it will return up to the maximum number of bytes in a packet and ignore all trailing null bytes. | IT Appendix |
| INITIALIZE | Prepares (formats) mass storage media for use by the computer. | Works the same as with BASIC/WS; however, only an *unmounted* disk may be INITIALIZE'd. | IM Chapter 2 |
| Interfaces not supported on BASIC/UX:<br>• BCD<br>• EPROM<br>• Powerfail<br>• Bubble Memory | Supported as stated in the BASIC/WS documentation. | These interfaces are not supported on BASIC/UX; however, you can gain access to their hardware by using the keywords READIO and WRITEIO. | Not available. |

| Keyword/Feature | BASIC/WS Description | BASIC/UX Description | For More Info |
|---|---|---|---|
| KNOBX<br>KNOBY | Return the net number of horizontal knob pulses (KNOBX) or vertical knob pulses (KNOBY). | Use BASIC/WS 3.0 definitions (and the KNB2_0 binary/definition is not supported). | IT<br>Chapter 11 |
| LIST | Lists the program currently in memory, and then shows the amount of memory available for program storage. | Lists the program, and then shows the sum of the memory available for program storage, stack, and COM. | UB<br>Chapter 2 |
| LIST BIN | Lists the name and version number of each binary currently in memory. | Functions the same as with BASIC/WS; however, the LIST BIN command will list all BASIC/WS binaries (the display will not vary since all binaries are always loaded). | IM<br>Chapter 5 |
| LOAD BIN | Loads binaries into memory. | Not supported by BASIC/UX, because all BASIC/WS binaries are part of the BASIC/UX system. | IM<br>Chapter 5 |
| ON/OFF CYCLE<br>ON/OFF DELAY<br>ON/OFF TIME<br>ON/OFF TIMEOUT | Enable/cancel event-initiated branches. | Work the same as with BASIC/WS; however, clock/timer resolution is 20 milliseconds (vs. 10 for BASIC/WS). A child process of BASIC/UX is spawned to handle timers. | PT<br>Chapter 9 |
| OUTPUT | Outputs items to a specified destination (device, I/O path, or string variable). | Works the same as with BASIC/WS; however, you can also output to a window and named or unnamed pipe. | IT<br>Chapter 4 |
| Memory Volume | BASIC/WS allows you to create a memory volume in your computer's random access memory (RAM). This provides a fast means for storing and retrieving files. | BASIC/UX will not support TRANSFERs to memory volumes. Also, BASIC/UX only supports LIF formatted memory volumes. It does not support HFS formatted memory volumes. | IM<br>Chapter 8 |

| Keyword/Feature | BASIC/WS Description | BASIC/UX Description | For More Info |
|---|---|---|---|
| msvs (mass storage volume specifier) | Allows you to specify mass storage devices other than the default volume. | Allows mapping of a volume specifier to an HP-UX directory name by using the specifier DISK in your BASIC/UX configuration file (*.rmbrc*). Also when the device type is HFS, the device selector is ignored unless it is mapped. Finally, `":HFS"` may be used to refer to the current working directory (the HFS directory that you had most recently moved to using the MASS STORAGE IS command). | IM Chapter 5 |
| PASS CONTROL | Passes Active Controller capability to a specified HP-IB device. | Passing control to an interface connected to a swap device or a mounted file system is *not* allowed. | IT Chapter 12 |
| PLOTTER IS<br><br>PRINTALL IS<br>PRINTER IS | PLOTTER IS selects the system plotting device.<br><br>PRINTALL IS selects a printing device for error messages, and PRINTER IS specifies the default system printing device (for PRINT, CAT, etc.). | These statements have been extended to allow:<br>• piping of output to an HP-UX command (for instance, to a spooler such as `"\| plt"`).<br>• redirection of output to a window.<br><br>Also, PRINTALL IS has been extended to include files. | GT Chapter 3 and PT Chapter 8 |
| PRINT LABEL | Gives a name to a mass storage volume. | Not supported on HFS volumes. | LR Keyword Dictionary |

| Keyword/Feature | BASIC/WS Description | BASIC/UX Description | For More Info |
|---|---|---|---|
| READ LABEL | Reads a volume label into a string variable. | Not supported on HFS volumes. | LR Keyword Dictionary |
| RE-SAVE | Creates an ASCII file and copies program lines as strings into that file. | BASIC/UX will create an HP-UX type file instead of an ASCII file when saving a program on an HFS volume. | LR Keyword Dictionary |
| "Run Light" and Other Activity Indicators | An asterisk (*) and other textual messages are shown in the lower, right-hand corner of the screen. | BASIC/UX provides: <br>• No Run Light indicator (this is on if KEY LABELS is off); <br>• A new activity indicator (**Execute**). | UB Chapter 1 |
| SAVE | Creates an ASCII file and copies program lines as strings into that file. | BASIC/UX will create an HP-UX type file instead of an ASCII file when saving a program on an HFS volume. | US Chapter 6 |
| SC | Returns the interface select code associated with an I/O path name. | Not supported for an I/O path name assigned to a file on an HFS volume. Also, it is not supported for HP-UX pipes. | IT Chapter 6 |
| SCRATCH BIN | Clears what SCRATCH A does, plus it clears all binaries except the CRT driver for the CRT in use. | All binaries are permanently loaded and therefore are non-scratchable. | UB Chapter 2 |
| SEPARATE ALPHA FROM GRAPHICS | Used to simulate the separate alpha and graphics rasters of Series 200 displays. | Same as the description for the keywords ALPHA ON and ALPHA OFF. | GT Chapter 2 |

| Keyword/Feature | BASIC/WS Description | BASIC/UX Description | For More Info |
|---|---|---|---|
| SET ALPHA MASK | Sets the alpha write-enable mask to a bit pattern | Same as the description for the keywords ALPHA ON and ALPHA OFF. | GT Chapter 2 |
| SET DISPLAY MASK | Sets the display write-enable mask to a bit pattern. | Same as the description for the keywords ALPHA ON and ALPHA OFF. | GT Chapter 2 |
| SET TIME | Resets the time-of-day given by the real-time clock. | Affects the "local" BASIC/UX clock value only (HP-UX system time and date will not change). It also allows re-synchronization with HP-UX time. | PT Chapter 9 |
| SET TIMEDATE | Resets the absolute seconds (time and date) given by the real-time clock. | Affects the "local" BASIC/UX clock value only (HP-UX system time and date will not change). It also allows re-synchronization with HP-UX time. | PT Chapter 9 |
| SRM Interface | Provides access to the HP Shared Resource Manager. | BASIC/UX only supports one SRM interface card per system. Also, the semantics for LOCK are different if more than one BASIC/UX process is running on the same system. Finally, support is only provided for SRM STATUS registers 3 and 6. | PT Chapter 7 |

| Keyword/Feature | BASIC/WS Description | BASIC/UX Description | For More Info |
|---|---|---|---|
| STATUS and CONTROL Registers:<br>• Datacomm<br>• RS-232 | Supported as described in the *BASIC Interfacing Techniques* manual. | Some registers are not supported or have slightly different definitions. | IT Chapter 13 and Chapter 14 |
| STORE SYSTEM | Stores the BASIC/WS operating system and all binaries currently in memory. | Not supported on BASIC/UX (not needed, since there are no separate binaries). | IM Chapter 5 |
| SYSBOOT | Returns control to the BOOT ROM to restart the system selection and configuration process. | Not supported on BASIC/UX. | IM Chapter 5 |
| SYSTEM$ ("AVAILABLE MEMORY") | Returns the amount of memory available for program storage. | Returns the sum of the memory available for program storage, stack, and COM. | UB Chapter 3 |
| SYSTEM$ ("MASS MEMORY") | Returns `X000YZ0000000000` where:<br>`X` = Number of internal disk drives;<br>`Y` = Number of initialized EPROM cards;<br>`Z` = Number of bubble memory cards.<br>(If `Y` or `Z` exceed 9, an asterisk appears.) | Always returns `0000000000000000`. | UB Chapter 3 |

| Keyword/Feature | BASIC/WS Description | BASIC/UX Description | For More Info |
|---|---|---|---|
| SYSTEM$ ("MASS STORAGE IS") | Returns the mass storage unit specifier of the current MASS STORAGE IS device as it appears in the CAT heading. | For HFS volumes, the device selector, unit, and volume numbers are not shown. For LIF and SRM volumes, it has not changed. | US Chapter 5 |
| SYSTEM$ ("SERIAL NUMBER") | Returns one of the following: • Bytes 4-14 of an ID PROM (if present); • an encoded serial number from an HP-HIL ID module (if present); otherwise, it returns the null string. | Returns an encoded serial number from an HP-HIL ID module (if present); (ID PROMs are only available with Series 200 computers); otherwise, it returns the null string. | UB Chapter 3 |
| TIMEDATE | Returns the current value of the real-time clock. | Works the same as with BASIC/WS; however, resolution is limited to 20 milliseconds (vs. 10 ms with BASIC/WS). | PT Chapter 9 |
| TIMEZONE IS | Specifies the offset from Greenwich Mean Time, providing compatibility with HP-UX time stamps on files when switching back and forth between the BASIC and HP-UX systems. | Works the same as on BASIC/WS; however, when used without a parameter the time zone is resynchronized with the HP-UX time zone. | PT Chapter 9 |

| Keyword/Feature | BASIC/WS Description | BASIC/UX Description | For More Info |
|---|---|---|---|
| TRANSFER | Initiates unformatted I/O transfers, which can take place concurrently with continued program execution. | Supports BASIC/WS TRANSFER-related statements:<br>• Memory-mapped I/O is selected if burst I/O is associated with an I/O path name.<br>• Concurrent inbound & outbound TRANSFERs are *not* permitted on an RS-232 interface.<br>• Runs as a separate process.<br>TRANSFER is only supported for HFS and not for SRM, LIF, or MEMORY volumes. | IT Chapter 9 |
| WAIT | Causes the computer to wait approximately the number of seconds specified before executing the next statement. | Works the same as on BASIC/WS; however, resolution is load dependent, but generally 60 milliseconds (vs. 10 ms on BASIC/WS). | PT Chapter 3 |

## BASIC/UX Mass Storage Differences

This section covers mass storage differences between BASIC/WS and BASIC/UX. Topics covered are:

- Maximum Number of Open HFS Files
- Locking SRM Files
- Leaving SRM Files Open
- SRM Security
- Using a Single SRM Interface Card with BASIC/UX
- Accessing LIF Media by a Single User

### Maximum Number of Open HFS Files

The HP-UX operating system imposes a limit on the maximum number of file descriptors that each process may use at any time (currently 60). BASIC/UX generally uses at least 10 of these internally, plus one for each device in use. Any time an HFS file is opened, another file descriptor is put into use until the file is closed. Note that opening a LIF or SRM file **does not** require the use of a file descriptor. If the maximum number of open files is reached, then some files must be closed. This may be done by executing `ASSIGN TO *`, pressing the Reset key, or executing `SCRATCH A`.

### Locking SRM Files

BASIC/UX allows several instances of HP BASIC to be run on the same system. However, the SRM server assumes that each process is running on a different system. Because of this, the semantics of LOCK are different if more than one BASIC/UX process is trying to access the same file at the same time. For example, an ENTER normally hangs until the file is unlocked. If the file is locked by another BASIC/UX process on the same system, however, the ENTER returns error 481.

### Leaving SRM Files Open

Because the BASIC/WS is a single user system, it can request that the SRM server close all files open for that system. This "clean up" operation is performed when the system boots, when a Reset key or SCRATCH A is done, etc. BASIC/UX, however, is a multi-user system, and such an action could cause the files of other users on the same system to be closed inappropriately. BASIC/UX attempts to close all files that it opens, however, there may occasionally be some left open for this reason. In general this is not a problem. These files may be closed by executing the "remove user" command on the SRM console, or running the hp-ux *srmclean*(1) utility. Be sure that no one else is using the SRM from that system before taking these actions.

### SRM Security

Because the hp-ux operating system allows multiple users to access the SRM from the same system, some access security is lost.

### Using a Single SRM Interface Card with BASIC/UX

The HP-UX operating system only supports one SRM card per system. This card can be shared simultaneously by several processes (BASIC/UX and/or the HP-UX SRM utilities), however, the performance will decrease as the number of processes using it increases.

### Accessing LIF Media by a Single User

It is highly recommended that only one process access a LIF media at a time. The HP-UX operating system coordinates multiple processes accessing the HFS file system simultaneously, and the SRM server does the same thing for the SRM file systems. However, there is no such support for multiple processes accessing the same LIF media simultaneously. For this reason, if more than one process does try to access the same LIF media at the same time, the media may become corrupt and data may be lost. Also, the accurate detection of media change can only be done when one process is accessing the media. Single process access is enforced for Amigo protocol devices. Once a process has begun accessing an Amigo protocol device, it "owns" that device until the Reset key is hit, a SCRATCH A is performed, or the process terminates.

## Overview of BASIC/UX "Enhancement" Features

Since BASIC/UX runs on the HP-UX operating system, it provides you with essentially all of the capabilities of HP-UX. Here are the categories of these new features:

- Multi-tasking (the ability to run several processes/programs concurrently — for example, support of HP terminals).
- Multi-user environment (the ability for several users to use the system simultaneously).
- Additional system status and configuration information (regarding processes, operating system, and windowing system).
- Ability to run HP-UX commands from within BASIC/UX.
- Networking capabilities (such as remote file access and network file transfer).
- Window management capabilities (X Windows).
- Additional interface control:
    - Interface mapping (for non-supported interfaces)
    - Interface locking.
    - Burst I/O.
- Use of HP-UX pipes.
- Selecting HP-HIL devices.
- HP-UX signal handling.
- HFS File buffering.
- Graphics buffering.

The following tables list the BASIC/UX enhancements to BASIC/WS. A manual name and chapter number are provided to help you locate more information on each feature.

## Legend for Subsequent Tables

After each brief keyword description, the table gives a manual name and chapter number to help you locate more information about the feature briefly described in the table. Here are the acronyms used in the tables to specify BASIC manuals.

| | |
|---|---|
| GT | *BASIC Graphics Techniques* |
| IM | *Installing and Maintaining the BASIC/UX System* |
| IT | *BASIC/UX Interfacing Techniques* |
| LR | *BASIC Language Reference* |
| PT | *BASIC Programming Techniques* |
| UB | *Using the BASIC/UX System* |

## Entering and Exiting the BASIC/UX Environment

Since the BASIC/UX system runs on the HP-UX operating system, you have the ability to enter BASIC/UX from an HP-UX shell prompt. (You can also have your System Administrator set up your console to automatically "wake up" in the BASIC/UX system. See the *Installing and Maintaining the BASIC/UX System* for details.)

| Keyword/Feature | Description | For More Info |
|---|---|---|
| rmb | Enters you into the BASIC/UX system (from an HP-UX shell). | UB Chapter 2 |
| QUIT | Exits you from the BASIC/UX system (returns you to an HP-UX shell). | UB Chapter 2 |

## Multi-Tasking and Multi-User Capabilities

You can run several BASIC/UX processes at the same time by executing the `rmb` command from within a windowing environment (while in an HP-UX shell). Each BASIC/UX process can be running a separate BASIC/UX program at the same time that other processes are running. (See *Using the BASIC/UX System* for instructions.)

You can also have several users running processes concurrently on the HP-UX system. Each one will be on a separate console or terminal. (See the HP-UX documentation for more details on this capability.)

## System Status and Configuration Information

The SYSTEM$ function has some additional capabilities with BASIC/UX.

| Keyword/Feature | Description | For More Info |
|---|---|---|
| SYSTEM$("PROCESS ID") | Returns the process identifier of the main BASIC/UX process. | UB Chapter 3 |
| SYSTEM$("VERSION:OS") | Returns the operating system version and name. | UB Chapter 3 |
| SYSTEM$ ("WINDOW SYSTEM") | Returns information that describes the windowing system currently in use. | UB Chapter 3 |

## Running HP-UX Commands from within BASIC/UX

You can send commands to HP-UX while in the BASIC/UX system. For instance, the following command returns a printout of a calendar for the current month:

```
EXECUTE "cal"
```

The results of the command appear in the HP-UX window from which BASIC/UX was started.

| Keyword/Feature | Description | For More Info |
|---|---|---|
| EXECUTE | Gives you access to the HP-UX operating system for executing commands and programs. | UB Chapter 9 |

## Window Management

A windowing system gives you the ability to partition your screen into several "panes", each of which is the display for a process. The BASIC/UX system uses the X Windows system. (The X Windows documentation provides a more complete description of a windowing system.)

| Keyword/Feature | Description | For More Info |
|---|---|---|
| CLEAR WINDOW | Clears the contents of the specified window. | UB Chapter 3; IT Chapter 17 |
| CONTROL CRT, 22 | Shuffles a specified window number to the top or bottom of the window stack. | LR Appendix C; IT Chapter 17 |
| CREATE WINDOW | Creates a window which can be accessed by BASIC/UX. | UB Chapter 3; IT Chapter 17 |
| DESTROY WINDOW | Deletes a window created by BASIC/UX. | UB Chapter 3; IT Chapter 17 |
| LIST WINDOW | Shows a list of all active BASIC/UX windows. | UB Chapter 3; IT Chapter 17 |
| MOVE WINDOW | Moves a window to another x-y location on the screen. | UB Chapter 3; IT Chapter 17 |
| SCRATCH WINDOW | Destroys all windows created from within the current BASIC/UX process. (Note that it will not remove the "boot" BASIC/UX window.) | UB Chapter 3; IT Chapter 17 |

## Additional Interface Control

The BASIC/UX system runs "on top of" the HP-UX multi-tasking, multi-user system. In this environment, several processes/users may attempt to access an interface simultaneously. Therefore, BASIC/UX provides a way of "locking" an interface to a process so that no other process(es) may use it.

BASIC/UX also has the capability of using HP-UX's "burst mode" of I/O operations. This mode is generally faster than the default (non-burst) mode of I/O operations used with BASIC/UX.

There are a few interface cards that are not supported by BASIC/UX such as the EPROM and BCD interface cards. To access these interface cards, you need to use memory-mapped I/O. Once one of these interfaces is memory mapped into the BASIC/UX address space, you can access locations on the interface using the keywords READIO and WRITIO.

The following keywords and registers provide you access to the above mentioned features:

| Keyword/Feature | Description | For More Info |
|---|---|---|
| CONTROL *Sel_code*, 255<br>STATUS *Sel_code*, 255 | CONTROL turns interface mapping, locking, and burst I/O on or off. STATUS queries the status of locking and burst mode for the interface at the specified select code. | IT<br>Chapter 12<br>and<br>Chapter 16 |

## Using HP-UX Pipes

The HP-UX system has a feature called a "pipe", which is a special memory buffer that is used to hold the output of one command so that it can be used as the input to another command. Pipes may be used with the following commands:

| Keyword/Feature | Description | For More Info |
|---|---|---|
| ASSIGN<br>CONTROL<br>DUMP DEVICE IS<br>ENTER<br>OUTPUT<br>PLOTTER IS<br>PRINTALL IS<br>PRINTER IS<br>STATUS<br>TRANSFER | These statements have been extended to allow piping of output to an HP-UX command (for example, to a spooler such as `"\| plt"`). | IT<br>Chap. 3<br>IT<br>Chap. 6<br>GT<br>Chap. 3<br>IT<br>Chap. 5<br>IT<br>Chap. 4<br>GT<br>Chap. 3<br>PT<br>Chap. 9<br>PT<br>Chap. 9<br>IT<br>Chap. 6<br>IT<br>Chap. 9 |

## Selecting HP-HIL Devices

With BASIC/UX, you have additional control capabilities for HP-HIL devices. You can enable or restrict a BASIC/UX process' access to selected HP-HIL devices by setting and clearing bits in a mask.

| Keyword/Feature | Description | For More Info |
|---|---|---|
| SET HIL MASK | Enables the specified HP-HIL devices to be used by the BASIC/UX system. (All other HP-HIL devices cannot be used by this particular BASIC/UX process.) | IT<br>Chapter 19 |

## Trapping HP-UX Signals

HP-UX processes use signals to communicate among one another. BASIC/UX can set up interrupt-branching to trap these signals.

| Keyword/Feature | Description | For More Info |
|---|---|---|
| ON EXT SIGNAL<br>OFF EXT SIGNAL | ON EXT SIGNAL defines an end-of-statement branch to be taken when an HP-UX signal is received. OFF EXT SIGNAL cancels the branch. | PT<br>Chapter 3 |
| STATUS 33, *Register* | Queries HP-UX system signal information. (The *Register* number corresponds to an HP-UX signal. See the *Language Reference* for a complete list of signals and register numbers.) | PT<br>Chapter 3<br>and<br>LR<br>Registers<br>Appendix |
| ENABLE EXT SIGNAL<br>DISABLE EXT SIGNAL | Enable and disable a specified HP-UX signal to cause interrupts (end-of-statement branches) or to cause default actions. | PT<br>Chapter 3 |

## HFS File Buffering

File buffering describes the way that the system handles data going to and coming from a disc.

| | |
|---|---|
| If file buffering is enabled: (default for BASIC/UX) | The system writes all file output data into a memory buffer. This buffer is then written onto the disc whenever it is filled, and at regular intervals (typically every 30 seconds) by the HP-UX *syncer* (1M) daemon. This buffering action reduces wear on the disc and speeds up most disc operations. |
| If file buffering is disabled: | All data output to the file is sent to the memory buffer then immediately to the disc. |

File buffering usually speeds up disc operations, but at the risk of losing the memory buffer's contents (if power were to fail before the buffer is flushed).

Note that file buffering is *only* available with HFS files. (LIF files are never buffered, and only the SRM controller dictates how buffering is handled with SRM files.)

| Keyword/Feature | Description | For More Info |
|---|---|---|
| CONTROL *IO_path*, 9<br>STATUS *IO_path*, 9 | CONTROL turns file I/O buffering on or off.<br>STATUS queries the status of file I/O buffering. | PT<br>Chapter 8 |

## Graphics Buffering

Graphics buffering describes the way that the system handles data going to and coming from the frame buffer. The following are GESCAPE operation selectors that help you perform graphics buffering:

| Keyword/Feature | Description | For More Info |
|---|---|---|
| GESCAPE | Operation selectors have the following definitions:<br><br>10 turns graphics buffering on;<br>11 turns graphics buffering off;<br>12 returns the current graphics buffering mode;<br>13 flushes the graphics buffer. | GT<br>Chapter 2 |

## BASIC/UX Mass Storage Enhancements

This section covers mass storage enhancements included with BASIC/UX. Topics covered are:

- Locking HFS Files
- Accessing Networked HFS File Systems
- Using Long File Names on the HFS File System.

### Locking HFS Files

The keywords LOCK and UNLOCK have been extended to work with HFS file systems. In general, the semantics are very similar to those of the SRM file system. For information on LOCKing and UNLOCKing HFS files, read the section "Locking HFS Files" in the chapter "Data Storage and Retrieval" found in the *BASIC Programming Techniques* manual.

### Accessing Networked HFS File Systems

BASIC/UX has been extended to allow access to networked HFS file systems through Remote File Access (RFA) and Network File Service (NFS). This allows files on other systems to be accessed in many of the same ways that local files are accessed. For information on networking with BASIC/UX, read the section "If You Have Networking Options" in the chapter "Using HP-UX Commands in BASIC/UX" found in the *Using the BASIC/UX System* manual.

### Using Long File Names on the HFS File Systems

The HP-UX operating system supports two types of HFS file systems. The Short File Name (SFN) system has a maximum file name length of 14 characters. Long File Name (LFN) systems, however, allow files names to be up to 255 characters long. BASIC/WS only supports SFN file systems, however, BASIC/UX will work with both types. See your HP-UX operating system documentation for details.

# Porting BASIC/WS Programs to BASIC/UX 20

# Porting BASIC/WS Programs to BASIC/UX 20

Most BASIC/WS programs will run on a BASIC/UX system with little or no modification. This chapter describes the process of copying, running, debugging, and modifying BASIC/WS programs for use on BASIC/UX.

## General Steps

1. Copy the BASIC/WS program to a disc that you are using for BASIC/UX. (This is a simple operation, since both systems can access HFS, LIF, and SRM discs.)
2. Run the program to find all *run-time* errors. As you encounter each error, note the BASIC keyword and look it up in the preceding chapter or in the *BASIC Language Reference.* Make the change, and re-run the program to find the next run-time error.
3. Once you have corrected all the run-time errors, you should store the program.
4. Run the program again to locate all *silent* errors (ones that do not generate a BASIC error message, but that do cause the program to work differently on BASIC/UX).
5. Once you have located and corrected all errors, store the program. This will complete the porting process.

## Prerequisite

If you are not already in BASIC/UX, you need to enter it before learning the above tasks. For information on how to do this, read the chapter called "Entering and Leaving BASIC/UX" in the *Using the BASIC/UX System* manual.

## Copying BASIC/WS Files to BASIC/UX

BASIC/WS programs can be stored in files on three possible disc formats:

- HFS (Hierarchical File System).
- LIF (Logical Interchange Format).
- SRM (Shared Resource Manager).

Since the BASIC/UX system also supports all three of these disc formats, you must decide whether to use the same disc or to use the HP-UX system's HFS disc. This section describes the necessary steps for copying files.

### Prerequisites

In order to use a LIF disc from BASIC/UX, the appropriate device files must be set up in your `/dev/rmb` directory. For information on how to do this, read the section "Setting Up LIF Devices" in the chapter "Maintaining the BASIC/UX System" in the *Installing and Maintaining the BASIC/UX System* manual.

### Copying Files from a LIF Disk

The following steps copy a file from a LIF disk to your current BASIC/UX working directory.

1. Catalog the files on the floppy using the CAT command. For example, the following command assumes you have a disk drive located at device selector 702 with unit 1.

   ```
   CAT ":,702,1"
   ```

2. Look at the catalog listing of the files on the disk and select the file you wish to COPY. For example, if you select file `Prog1`, execute the following command to copy the file into your current BASIC/UX working directory:

   ```
   COPY "Prog1:,702,1" TO "Prog1"
   ```

   This file can now be loaded as explained in the section "Debugging the Program."

## Copying Files from an SRM Disk

The following are steps required to copy a file from the SRM file system into the current working directory of your BASIC/UX system. (These steps assume your BASIC/UX system is connected to an SRM system through an HP 98629A or HP 50961A interface card located in your computer's backplane.) The steps are as follows:

1. Catalog the files on the SRM file system using the CAT command. For example, the following command assumes the SRM interface is located at select code 8 and the file you are looking for is located in the root-level "USERS" directory on the server with node number 1.

   ```
   CAT "/USERS:REMOTE 8,1"
   ```

2. Look at the cataloged listing of the files on the disk and select the file you wish to COPY. For example, if you select a file `/USERS/Prog1`, execute the following command to copy the file into your current BASIC/UX working directory:

   ```
   COPY "/USERS/Prog1:REMOTE 8,1" TO "Prog1"
   ```

   This file can now be loaded into memory as explained the the section "Debugging the Program."

## Copying Files from a Mounted HFS File System

The following are steps required to copy a file from a mounted HFS disk into the current working directory of your BASIC/UX system. (These steps assume the HFS disk has been "mounted" in the HP-UX system. If not, see your System Administrator.)

The steps are as follows:

1. Catalog the files on the HFS disk using the CAT command. For example, the following command assumes the HFS disk is mounted at the location of the "/users" directory.

   ```
   CAT "/users"
   ```

2. The following `COPY` command copies the file named "Prog1" located in the root-level "users" directory:

   ```
   COPY "/users/Prog1:HFS" TO "Prog1:HFS"
   ```

   where "Prog1" is the name of the new copy of the file. (The files can have the same name if they are not in the same directory.) Note that if your current MASS STORAGE IS directory is on HFS, you may omit the `":HFS"` portion of the above file specifiers.

## Debugging the Program

### Overview

This section explains how to load, run, debug, and correct your BASIC/WS programs.

### Prerequisites

You should have already copied all BASIC/WS programs into your current BASIC/WS working directory.

### Loading the Program

To load a program, use either the GET or LOAD command. The GET command allows you to load ASCII and HP-UX files, and the LOAD command allows you to load PROG files. To determine whether the file you are loading is an ASCII or a PROG file, execute the CAT command. This command will list the TYPE attribute for the files in your current directory or the directory that you specified when you executed the CAT command. Once you have determined the type of file you are to load, you can then select the appropriate command for loading that file. For example, assuming there is a file named `Prog1` in the directory you are currently in, you would execute this command:

```
GET "Prog1"      Get an "ASCII" or "HP-UX" type file.
```

or

```
LOAD "Prog1"     Load a "PROG" type file.
```

Note that you can MSI to any file system (LIF, SRM, or mounted HFS) and use the above commands to load a file. For more information on loading programs read the chapter "Loading and Running Programs" found in the *Using the BASIC/UX System* manual.

### Running the Program

To run a program that has been loaded into memory, use the RUN command. For example, type:

```
RUN [Return]
```

or press the softkey labeled **RUN**.

Subsequent pages describe how to correct errors in the program.

## Recognizing and Correcting Run-Time Errors

If the BASIC/UX system should encounter an error while the program is running (a "run-time" error), it will report an error message containing the line number where the error occurred. For example:

```
ERROR 150 IN 110   Bad select code or device spec
```

where `150` is the error number, `110` is the line number where the error occurred (in some cases it is the approximate line number), and `Bad select code or device spec` is the error message. These items are important in correcting your program's run-time errors. It would be a good idea to write this information down as well as the statement that you feel caused the error.

The steps you need to take toward correcting your run-time errors are as follows:

1. Look up the error number in the appendix entitled "Error Messages" in the *BASIC Language Reference*. This will in some cases give you a more complete error message and better help you determine what is wrong.
2. Use the statement's keyword to direct you to the location in the "Summary of Keyword Differences" chapter that will help you in most cases find a solution to the run-time error. For example, if the run-time error is an I/O error dealing with the TRANSFER statement, then you would look for the TRANSFER statement in the "Summary of Keyword Differences" table in the preceding chapter. This table will give you a comparison of how this keyword is used on BASIC/WS and BASIC/UX. It also refers to a more detailed tutorial in one of the *Techniques* manuals. (Note that the *BASIC Language Reference* contains the most detailed information on each BASIC keyword.
3. Once you have found what your error is, correct it and re-run the program. You should continue to do this until you have corrected all the run-time errors in your program. When the errors have all been corrected, you should make sure that there are no "silent" errors in the program. Silent errors are differences in the way the program *actually* runs now and how it *should* run (see the next section).
4. When there are no more run-time errors in your program, you should RE-STORE or RE-SAVE it.

*(Continued on next page)*

## Recognizing and Correcting Silent Errors

Silent errors are differences between:

- The way the program *should* run, and
- The way it *actually* runs on BASIC/UX.

They do *not* generate an error message, as is the case with run-time errors.

For example, the color of your graphics output might have changed. This is not an error that would cause the program to stop working; however, it is an error and it must be corrected to make the program function as it did prior to running it on BASIC/UX.

The steps to take toward correcting your silent errors are as follows:

1. Locate the places in the program that exhibit behavior different from the way the program used to work.
2. Isolate the silent error to a certain part of your program by:
   - Reading the program listing.
   - Single-stepping through the program using the `Step` key ($\boxed{\text{f1}}$ in the System menu of your ITF keyboard).
   - Using the `Pause` key where unusual behavior occurs.

   The procedures for using these statements to troubleshoot a program are described in the chapter called "Debugging Programs" found in the *BASIC Programming Techniques* manual. Following the procedures explained in these sections will help you find the statement that is causing your silent error.
3. Use the statement's keyword to direct you to the location in the "BASIC/UX Differences and Enhancements" chapter that helps you, in most cases, find a solution to the silent error. For example, if the silent error is a graphics error dealing with the SEPARATE ALPHA FROM GRAPHICS statement, then you would look for this statement in the "Summary of Keyword Differences" table. The table gives you a comparison of how this keyword is different from BASIC/WS to BASIC/UX. It also provides a reference to more detailed information. Continue debugging until you have corrected all errors in your program.
4. When the errors have all been corrected, RE-STORE or RE-SAVE the program. It is ready for use on BASIC/UX.

# Index

HEWLETT
PACKARD

**HP Part Number**
**98796-90010**

Microfiche No. 98796-99010
Printed in U.S.A.   E1188

98796-90610
For Internal Use Only