BASIC 5.0/5.1 Programming Techniques

Vol. 2: Porting Information

HP 9000 Series 200/300 Computers

HP Part Number 98613-90012

HEWLETT PACKARD

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

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

Printing History

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

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

January 1987...Edition 1

November 1987...Edition 2. This edition reflects 5.0 corrections and 5.1 additions.

Table of Contents

Chapter 1: Manual Organization
- Welcome 1-1
- What's In This Manual? 1-2
  - Programming Techniques Volume 1 1-2
  - Programming Techniques Volume 2 1-5
  - What's Not in this Manual 1-5

Chapter 2: Program Structure and Flow
- The Program Counter 2-2
- Sequence 2-3
  - Linear Flow 2-3
  - Halting Program Execution 2-3
  - Simple Branching 2-6
  - Using GOSUB 2-7
- Selection 2-10
  - Conditional Execution of One Segment 2-11
  - Choosing One of Two Segments 2-15
  - Choosing One of Many Segments 2-17
- Repetition 2-22
  - Fixed Number of Iterations 2-22
  - Conditional Number of Iterations 2-25
  - Arbitrary Exit Points 2-27
- Event-Initiated Branching 2-30
  - Types of Events 2-31
  - Example of Event-Initiated Branching 2-32
  - Example of Using the Knob 2-34
  - Deactivating Events 2-36
  - Disabling Events 2-38
- Chaining Programs 2-39
  - Using LOAD 2-39
  - Using GET 2-40
  - Program-to-Program Communications 2-41

Chapter 3: Numeric Computation

Numeric Data Types 3-1
REAL Data Type 3-1
INTEGER Data Type 3-2
COMPLEX Data Type 3-2
Declaring Variables 3-3
Assigning Variables 3-4
Implicit Type Conversions 3-4
Precision and Accuracy: The Machine Limits 3-7
Internal Numeric Formats 3-8
Evaluating Scalar Expressions 3-9
The Hierarchy 3-9
The Delayed Assignment Surprise 3-12
Operators 3-12
Numerical Functions 3-15
Arithmetic Functions 3-15
Array Functions 3-16
Exponential Functions 3-17
Trigonometric Functions 3-17
Hyperbolic Functions 3-18
Binary Functions 3-18
Limit Functions 3-19
Rounding Functions 3-19
Random Number Function 3-22
Complex Functions 3-22
Time and Date Functions 3-29
Base Conversion Functions 3-30
General Functions 3-31

Chapter 4: Numeric Arrays

Dimensioning an Array 4-2
Some Examples of Arrays 4-3
Problems with Implicit Dimensioning 4-8
Finding Out the Dimensions of an Array 4-9
Using Individual Array Elements 4-10
Assigning an Individual Array Element 4-10
Extracting Single Values From Arrays 4-10
Filling Arrays 4-11
Assigning Every Element in an Array the Same Value 4-11
Using the READ Statement to Fill an Entire Array 4-11
Copying Entire Arrays into Other Arrays 4-12

Printing Arrays 4-14
Printing an Entire Array 4-14
Examples of Formatting Arrays for Display 4-14
Passing Entire Arrays 4-16
Copying Subarrays 4-16
Redimensioning Arrays 4-24
Arrays and Arithmetic Operators 4-26
Using the MAT Statement 4-26
Performing Arithmetic Operations with Complex Arrays 4-28
Summing the Elements in an Array 4-29
Boolean Arrays 4-30
Reordering Arrays 4-32
Sorting Arrays 4-34
Sorting with Automatic REORDER 4-34
Sorting to a Vector 4-36
Searching Numeric Arrays 4-38
Numeric Comparisons in MAT SEARCH 4-42
Matrices and Vectors 4-47
Matrix Multiplication 4-47
Multiplication With Vectors 4-48
Identity Matrix 4-52
Inverse Matrix 4-53
Solving Simultaneous Equations 4-55
Singular Matrices 4-57
The Determinant of a Matrix 4-59
Ill-Conditioned Matrices 4-60
Detecting Ill-conditioned Matrices 4-61
Miscellaneous Matrix Functions 4-63
Transpose Function 4-63
Summing Rows and Columns of a Matrix 4-65
Examples of Complex Array Operations 4-67
Using Arrays for Code Conversion 4-70

Chapter 5: String Manipulation
String Storage 5-2
String Arrays 5-3
Evaluating Expressions Containing Strings 5-4
Evaluation Hierarchy 5-4
String Concatenation 5-4
Relational Operations 5-5

Substrings 5-6
Single-Subscript Substrings 5-6
Double-Subscript Substrings 5-7
Special Considerations 5-8
String-Related Functions 5-10
Current String Length 5-10
Maximum String Length 5-10
Substring Position 5-11
String-to-Numeric Conversion 5-13
Numeric-to-String Conversion 5-15
CRT Character Set 5-16
String Functions 5-17
String Reverse 5-17
String Repeat 5-18
Trimming a String 5-19
Case Conversion 5-19
Copying String Arrays and Subarrays 5-21
Searching and Sorting 5-22
Reordering an Array 5-28
Searching for Strings 5-30
Searching String Arrays 5-32
Number-Base Conversion 5-34
Introduction to Lexical Order 5-36
Why Lexical Order? 5-36
How It Works 5-36
The ASCII Character Set 5-37
The Extended Character Set 5-41
Predefined Lexical Order 5-43
Lexical Tables 5-45
Notation 5-45
ASCII Lexical Order 5-46
Case Conversions 5-46
FRENCH Lexical Order 5-48
Case Conversions 5-48
GERMAN Lexical Order 5-50
Case Conversions 5-50
SPANISH Lexical Order 5-52
Case Conversions 5-52
SWEDISH Lexical Order 5-54
Case Conversions 5-54
User-defined LEXICAL ORDER 5-56

User-Defined Lexical Orders 5-58
Sequence Numbers 5-60
Mode Entries 5-60
Bits, Bytes, and Mode Types 5-61

Chapter 6: Subprograms and User-Defined Functions

Some Examples 6-1
Benefits of Subprograms 6-3
A Closer Look at Subprograms 6-4
Calling and Executing a Subprogram 6-4
Differences Between Subprograms and Subroutines 6-5
Subprogram Location 6-5
Subprogram and User-Defined Function Names 6-5
Difference Between a User-Defined Function and a Subprogram 6-6
Numeric Functions and String Functions 6-6
Program/Subprogram Communication 6-8
Parameter Lists 6-8
Passing By Value vs. Passing By Reference 6-9
OPTIONAL Parameters 6-12
COM Blocks 6-14
Context Switching 6-18
Variable Initialization 6-19
Subprograms and Softkeys 6-19
Subprograms and the RECOVER Statement 6-20
Calling Subprograms from the Keyboard 6-21
Speed Considerations 6-21
Using Subprogram Libraries 6-23
Why Use Subprogram Libraries? 6-23
Listing the Subprograms in a PROG File 6-23
Loading Subprograms 6-23
Deleting Subprograms 6-26
Editing Subprograms 6-27
SUBEND and FNEND 6-29
Recursion 6-30
Top-Down Design 6-32
The Problem 6-32
A Data Structure 6-33

Chapter 7: Data Storage and Retrieval

Storing Data in Programs 7-2
Storing Data in Variables 7-2
Data Input by the User 7-2
Using DATA and READ Statements 7-3
File Input and Output (I/O) 7-7
Brief Comparison of Available File Types 7-7
Overview of File I/O 7-9
A Closer Look at General File Access 7-11
A Closer Look at Using ASCII Files 7-16
Example of ASCII File I/O 7-16
Data Representations in ASCII Files 7-17
Formatted OUTPUT with ASCII Files 7-19
Formatted ENTER with ASCII Files 7-24
A Closer Look at BDAT and HP-UX Files 7-26
Data Representations Available 7-26
Random vs. Serial Access 7-26
Data Representations Used in BDAT Files 7-27
Data Representations with HP-UX Files 7-30
BDAT File System Sector 7-31
Defined Records 7-31
EOF Pointers 7-35
Writing Data 7-37
Serial OUTPUT 7-37
Random OUTPUT 7-42
Reading Data From BDAT and HP-UX Files 7-46
Accessing Files with Single-Byte Records 7-49
Trapping EOF and EOR Conditions 7-50
Extended Access of Directories 7-53
Cataloging Individual PROG Files 7-53
Cataloging to a String Array 7-55
Getting an "Extended" Catalog of a LIF or HFS Disc 7-57
Getting a Count of Files Cataloged 7-57
Suppressing the Catalog Header 7-58
Cataloging Selected Files 7-58
Getting a Count of Selected Files 7-60
Skipping Selected Files 7-61

Chapter 8: Using a Printer

Printers Supported 8-1
Installing, Configuring, and Verifying Your Printer 8-2
The System Printer 8-2

Device Selectors 8-3
Primary Addresses 8-4
Using Device Selectors 8-6
Using the External Printer 8-7
Control Characters 8-7
Escape-Code Sequences 8-8
Formatted Printing 8-9
Using Images 8-11
Special Considerations 8-16
Using SRM Printers through the Spooler 8-17
Using a Spooler 8-17

Chapter 9: The Real-Time Clock
Initial Clock Value 9-1
Do You Have a Non-Volatile Clock? 9-2
Clock Range and Accuracy 9-2
Reading the Clock 9-3
Determining the Date and Time of Day 9-3
Setting the Clock 9-4
Clock Time Format 9-4
Setting Only the Time 9-6
Setting Only the Date 9-8
Using Clock Functions and Example Programs 9-11
Day of the Week 9-13
Days Between Two Dates 9-13
Interval Timing 9-13
Branching on Clock Events 9-14
Cycles and Delays 9-15
Time of Day 9-16
Priority Restrictions 9-17
Branching Restrictions 9-18

Chapter 10: Communicating with the Operator
Overview of Human I/O Mechanisms 10-2
Other Factors 10-2
Displaying and Prompting 10-3
Displaying Messages: A Two-Step Process 10-3
Turning Off Unwanted Modes 10-3
Clearing the Screen 10-5
Determining Screen Width and Height 10-7
Changing Alpha Height 10-8
Displaying Characters on the Screen 10-8

Custom Character Fonts 10-9
Character Cells 10-9
Example of Changing One Character 10-13
Editing Supplied Fonts 10-14
Generating Sound 10-16
Operator Input 10-24
Softkey Inputs 10-24
Using Knobs 10-32
Using Control Dials 10-33
Accepting Alphanumeric Input 10-36
Get Past the First Trap 10-36
Entering a Single Item 10-38
LINPUT with Multiple Fields 10-41
Yes and No Questions 10-43
Example Human Interfaces 10-45
An Expanded Softkey Menu 10-45
Moving a Pointer 10-51
An Example Custom Keyboard Interface 10-54

Chapter 11: Handling Errors
Overview of Error Responses 11-1
Anticipating Operator Errors 11-2
Boundary Conditions 11-2
REAL and COMPLEX Numbers and Comparisons 11-3
Trapping Errors with BASIC Programs 11-5
Setting Up Error Service Routines (ON/OFF ERROR) 11-5
Disabling Error Trapping (OFF ERROR) 11-6
Determining Error Number and Location (ERRN, ERRLN, ERRL, ERRDS, ERRM$) 11-7
A Closer Look at ON ERROR GOSUB 11-8
A Closer Look At ON ERROR GOTO 11-9
A Closer Look At ON ERROR CALL 11-11
A Closer Look At ON ERROR RECOVER 11-13
Simulating Errors (CAUSE ERROR) 11-14
Clearing Error Conditions (CLEAR ERROR) 11-16

Chapter 12: Debugging Programs
Using Live Keyboard 12-2
Executing Commands While a Program Is Running 12-2
Using Program Variables 12-2
Calling Subprograms 12-4

Pausing and Continuing a Program 12-5
Keyboard Commands Disallowed During Program Execution 12-5
Cross References 12-6
Generating a Cross-Reference Listing 12-6
Single-Stepping a Program 12-10
Tracing 12-12
TRACE ALL 12-12
PRINTALL IS 12-14
TRACE PAUSE 12-15
TRACE OFF 12-16
The CLR I/O (Break) Key 12-16

Chapter 13: Efficient Use of the Computer's Resources
Data Storage 13-1
Data Storage in Read/Write Memory 13-1
Data Storage on Mass Memory Devices 13-3
Comments and Multi-character Identifiers 13-4
Variable and Array Initialization 13-4
Mass Memory Performance 13-5
Program Files 13-5
Data Files 13-6
Benchmarking Techniques 13-7
INTEGER Variables 13-9
Minimum and Maximum Values 13-9
Mathematical Operations 13-9
Loops 13-11
Array Indexing 13-12
REAL and COMPLEX Numbers 13-13
Minimum and Maximum Values 13-13
Type Conversions 13-13
Constants 13-14
Polynomial Evaluations 13-14
Logical Comparisons for Equality on REAL Numbers 13-17
Saving Time 13-18
Multiply vs. Add 13-18
Exponentiation vs. Multiply and SQRT 13-18
Array Fetches vs. Simple Variables 13-19
Concatenation vs. Substring Placement 13-19
HP 98635 Floating-Point Math Card 13-21
MC68881 Floating-Point Math Co-Processor 13-21
Enabling and Disabling Floating-Point Math Hardware 13-21
MC68020 Internal Cache Memory 13-22

Saving Memory 13-23
Releasing Memory Volumes 13-25

Chapter 14: Porting to 3.0
Porting Topics Covered 14-2
Compatibility with Preceding Versions 14-3
Configuring BASIC 14-3
Helpful Documentation 14-3
Missing Language Extensions BIN Files 14-4
Missing Driver BIN Files 14-5
Statement Changes 14-6
CSUBs 14-6
PHYREC 14-7
Knob 14-9
The KNOBX Function 14-9
Keyboards with Built-in Knob 14-10
HP-HIL Keyboards with Mouse 14-11
Programming for Both Versions and Keyboards 14-12
KNB2_0 14-13
Graphics 14-14
Default Plotter 14-14
Implicit GCLEAR 14-14
Input Device Viewport 14-14
Graphics Tablet DIGITIZE 14-15
The VIEWPORT Statement 14-15
The PIVOT Statement 14-19
Display Functions 14-25
Prerun On LOADSUB 14-26
Special Case of I/O Transfers 14-26

Chapter 15: Porting to Series 300 and 4.0
Methods of Porting 15-2
Chapter Organization 15-2
Description of Series 300 Hardware 15-3
Displays 15-4
Processor Boards 15-6
Battery-Backed Real-Time Clock 15-6
Built-In Interfaces 15-7
ID PROM 15-10
Just Loading and Running Programs 15-11
Should Problems Arise 15-11

Using a Configuration Program 15-12
HP 98644 Serial Interface Configuration 15-12
HP 98203 Keyboard Compatibility Mode 15-13
Configuring Separate Alpha and Graphics Planes 15-27
Using the Display Compatibility Interface 15-28
Hardware Description 15-29
Steps in Using this Card Set 15-31
Switching Back to the Series 300 Display 15-32
Automatic Display Selection at System Boot 15-33
Removing Display Drivers 15-33
If Your Screen Is Blank 15-33
Modifying the Source Program (Porting to 4.0) 15-35
Incompatible CSUBs 15-35
HP 98203 Specific Key Codes 15-35
Additional Porting Considerations 15-36
BASIC 4.0 Enhancements for Series 200 Computers 15-40

Chapter 16: Porting to 5.0
Compatibility with Previous Versions 16-1
Categories of New Features 16-2
New Hardware Supported 16-2
New Utilities 16-3
HFS Disc Support 16-4
Human Interface Enhancements 16-5
New Keywords that Duplicate Register Operations 16-6
General Programming Additions 16-7
New STATUS/CONTROL Registers 16-8
Additional HP-HIL Support 16-9
Additional Graphics Features 16-10
Additional CSUB Capabilities 16-11

Chapter 17: Porting and Sharing Files
Sharing HFS Discs and Data Files 17-2
General Compatibility Requirements 17-2
Common File Types 17-3
Common Data Types 17-4
HP-UX Text and Binary Files 17-5
Examples of HP-UX File Access: Textual Numeric Data 17-6
Porting LIF Files to SRM 17-24
SRM File Specifiers 17-24
SRM Mass Storage Volume Specification 17-25
Allowing for SRM Directory Paths 17-26

SRM Passwords vs. LIF Protect Codes 17-27
Copying Item-by-Item Using ENTER and OUTPUT 17-28
Accessing Files Created on Non-Series-200/300 SRM Workstations 17-29

Chapter 18: 5.1 Enhancements
Functionality Additions 18-1
Manual Changes 18-1

Index

Porting to 3.0 14

Porting Topics Covered 14-2
Compatibility with Preceding Versions 14-3
Configuring BASIC 14-3
Helpful Documentation 14-3
Missing Language Extensions BIN Files 14-4
Missing Driver BIN Files 14-5
Statement Changes 14-6
CSUBs 14-6
PHYREC 14-7
Knob 14-9
The KNOBX Function 14-9
Keyboards with Built-in Knob 14-10
HP-HIL Keyboards with Mouse 14-11
Programming for Both Versions and Keyboards 14-12
KNB2_0 14-13
Graphics 14-14
Default Plotter 14-14
Implicit GCLEAR 14-14
Input Device Viewport 14-14
Graphics Tablet DIGITIZE 14-15
The VIEWPORT Statement 14-15
The PIVOT Statement 14-19
Display Functions 14-25
Prerun On LOADSUB 14-26
Special Case of I/O Transfers 14-26

14 Porting to 3.0

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.

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
```

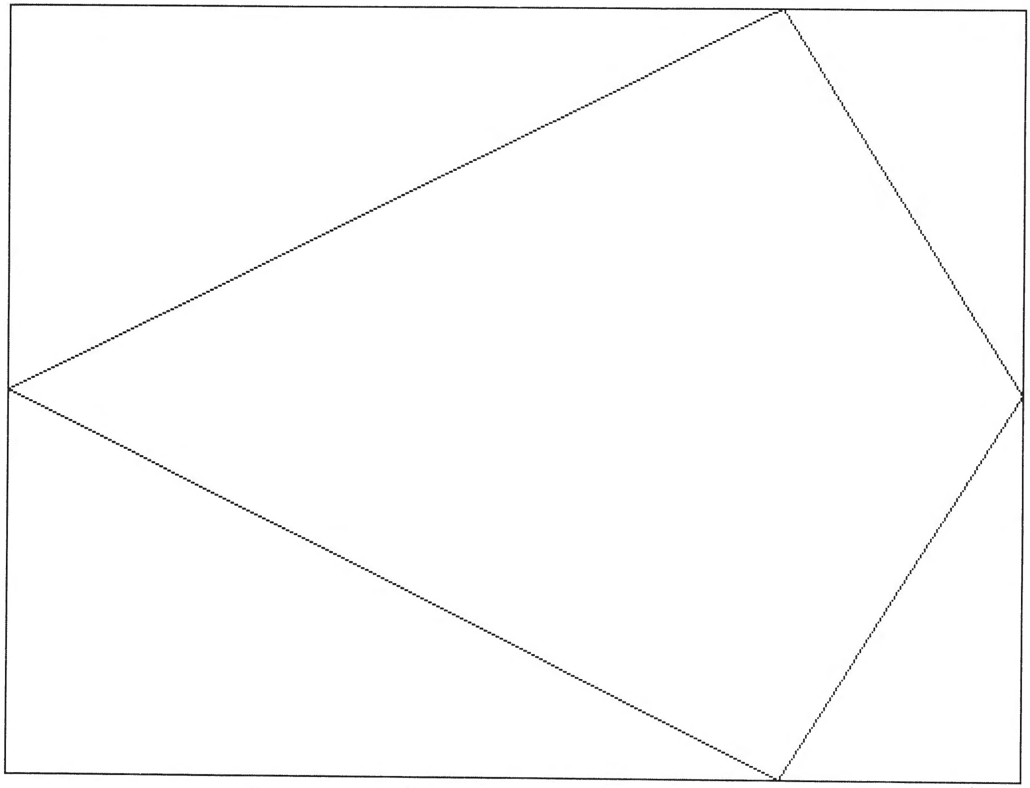


Figure 14-1. BASIC 2.0/2.1 and 3.0 without VIEWPORT

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.

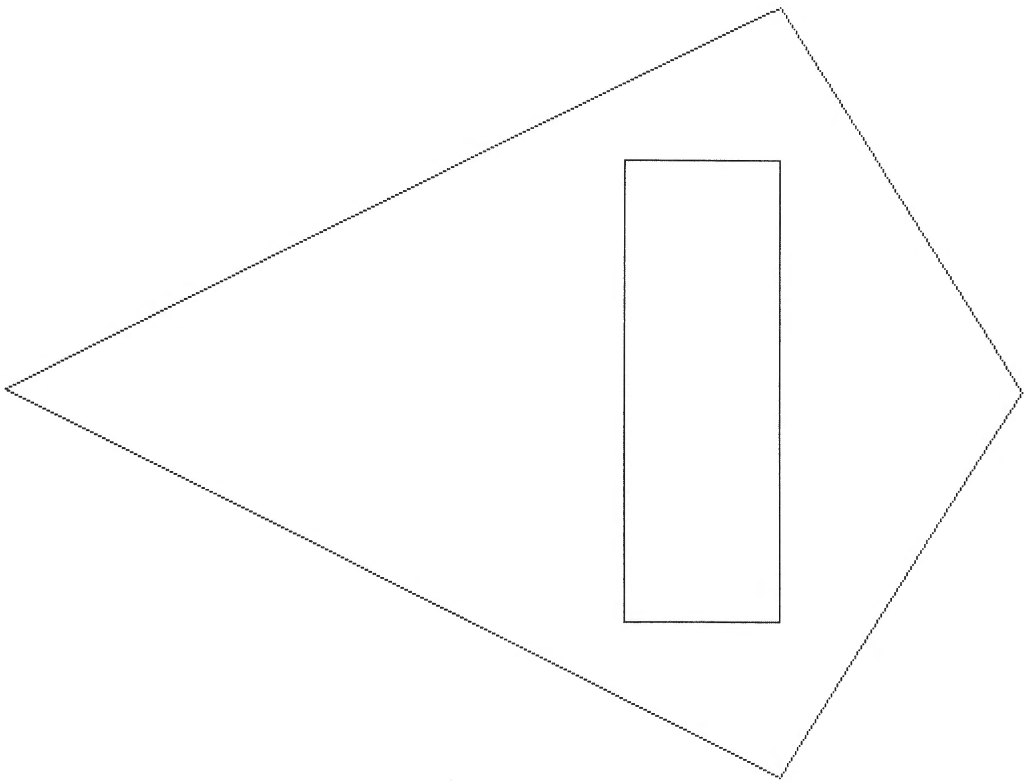


Figure 14-2. BASIC 2.0/2.1 with VIEWPORT

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.

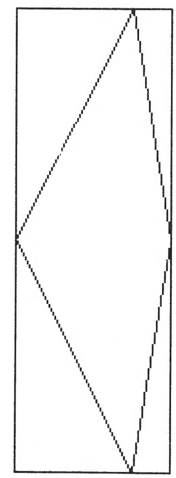


Figure 14-3. BASIC 3.0 with VIEWPORT

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
```

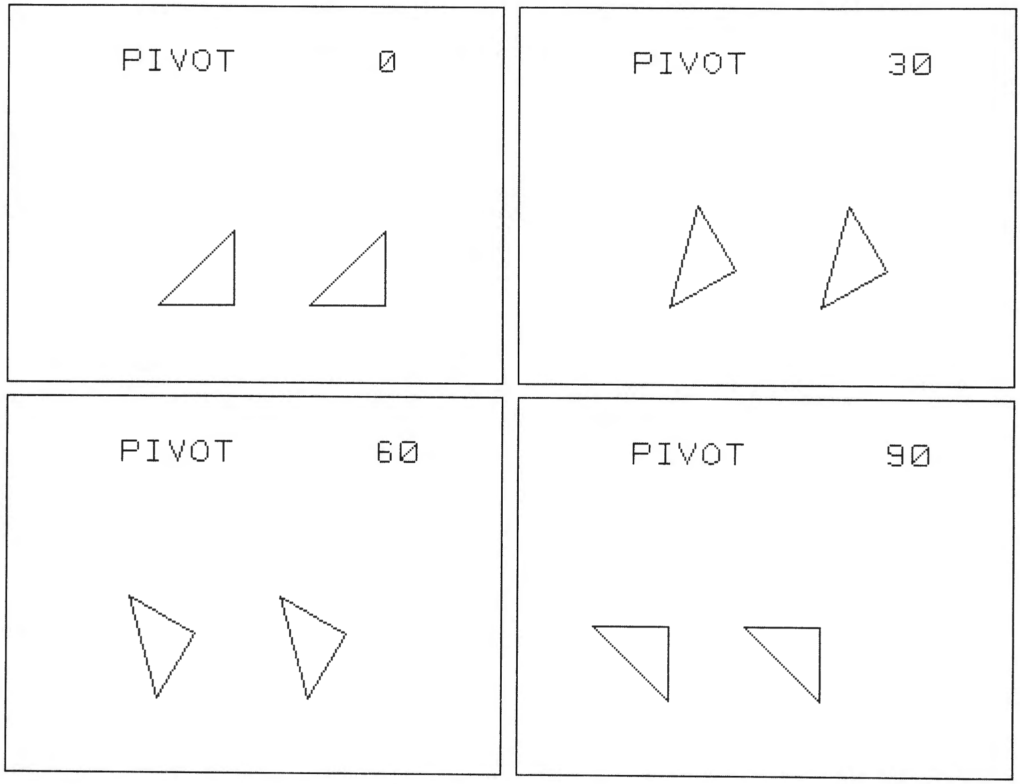


Figure 14-4. BASIC 2.0/2.1 RPLOT with PIVOT

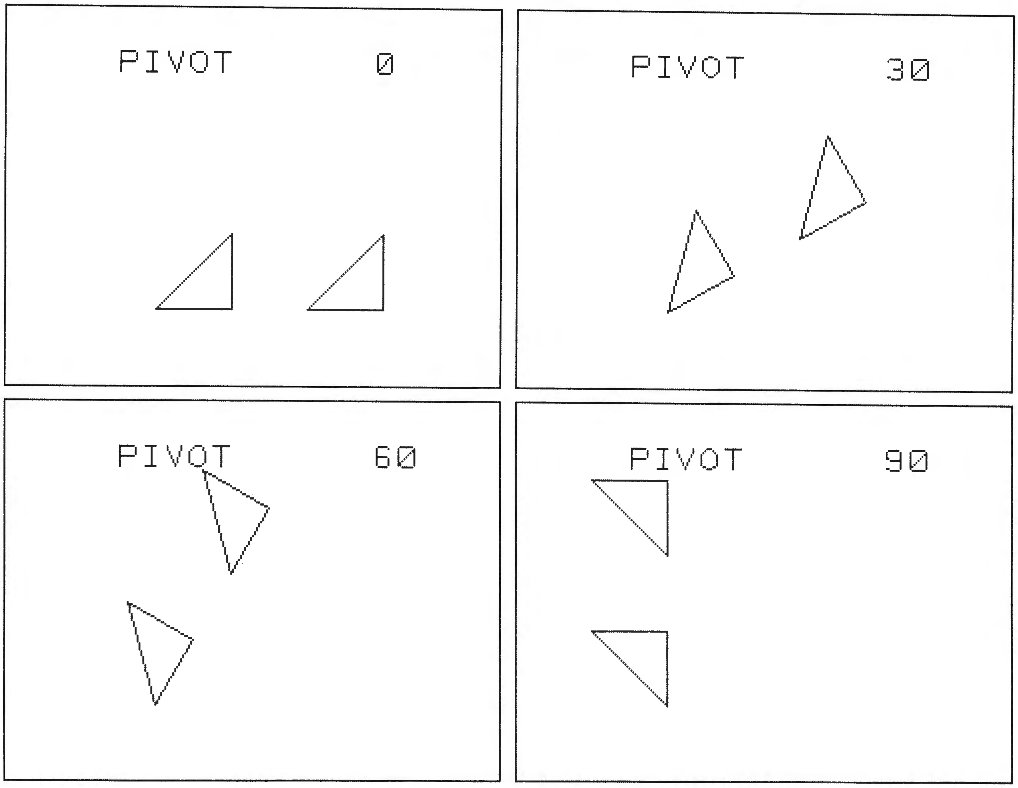


Figure 14-5. BASIC 3.0 RPLOT with PIVOT

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
```

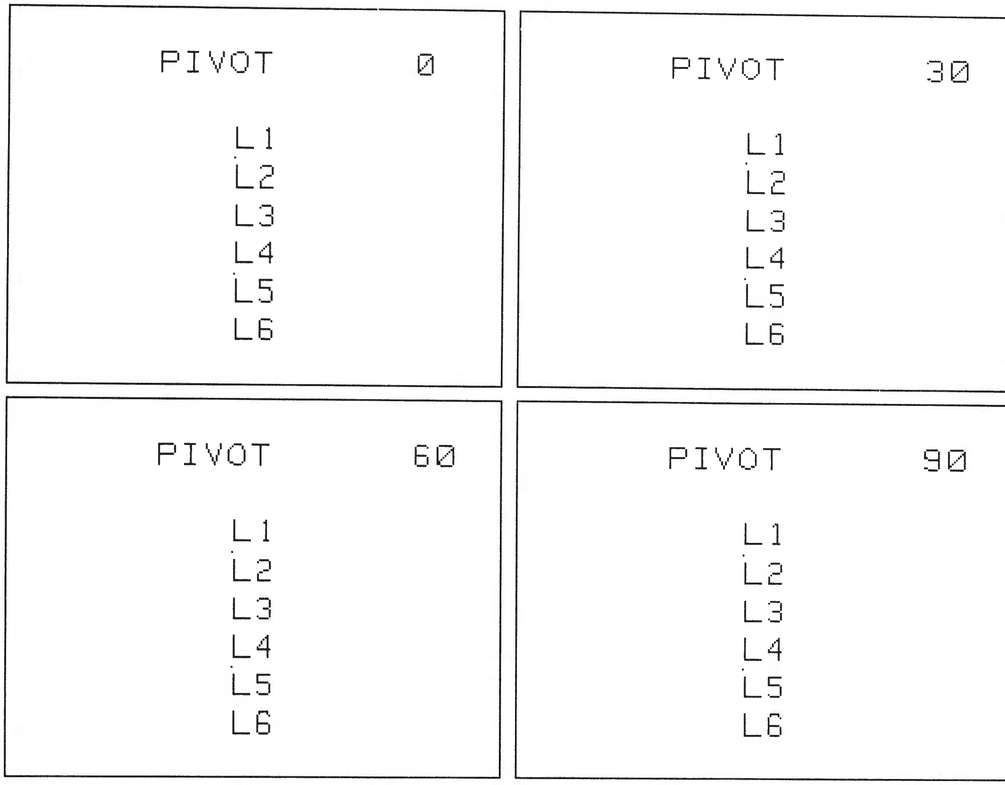


Figure 14-6. BASIC 2.0/2.1 LABEL with PIVOT

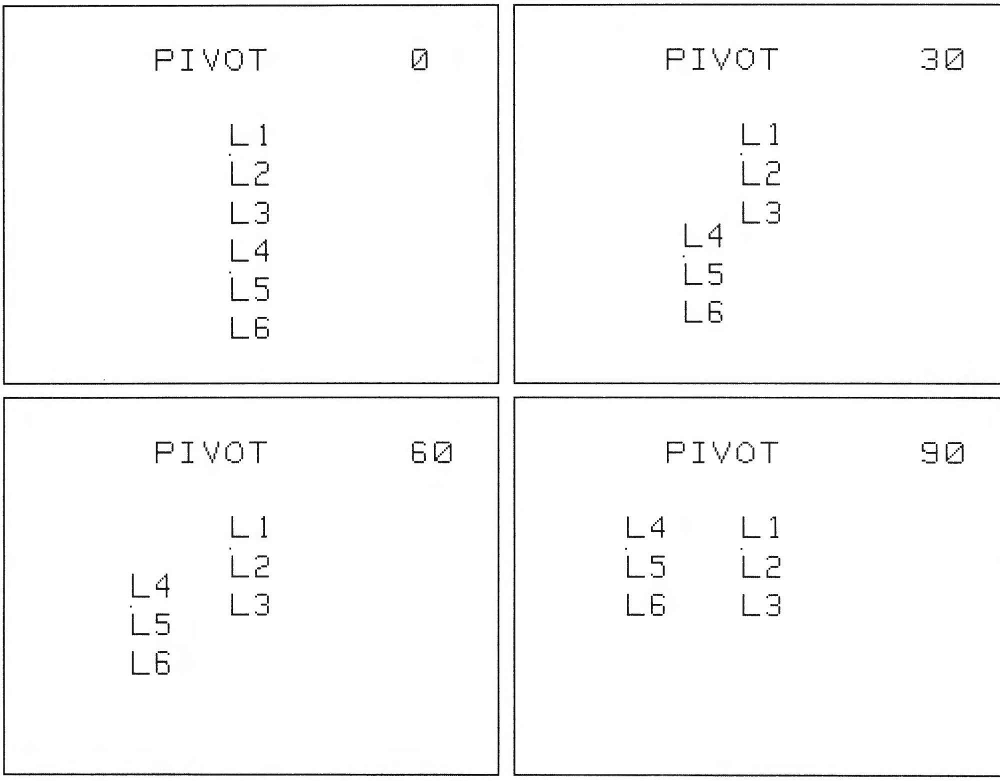


Figure 14-7. BASIC 3.0 LABEL with PIVOT

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:

```
hp HI THERE hp CR
LF
```

The hp 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

Methods of Porting 15-2
Chapter Organization 15-2
Description of Series 300 Hardware 15-3
Displays 15-4
Processor Boards 15-6
Battery-Backed Real-Time Clock 15-6
Built-In Interfaces 15-7
ID PROM 15-10
Just Loading and Running Programs 15-11
Should Problems Arise 15-11
Using a Configuration Program 15-12
HP 98644 Serial Interface Configuration 15-12
HP 98203 Keyboard Compatibility Mode 15-13
Configuring Separate Alpha and Graphics Planes 15-27
Using the Display Compatibility Interface 15-28
Hardware Description 15-29
Steps in Using this Card Set 15-31
Switching Back to the Series 300 Display 15-32
Automatic Display Selection at System Boot 15-33
Removing Display Drivers 15-33
If Your Screen Is Blank 15-33
Modifying the Source Program (Porting to 4.0) 15-35
Incompatible CSUBs 15-35
HP 98203 Specific Key Codes 15-35
Additional Porting Considerations 15-36
BASIC 4.0 Enhancements for Series 200 Computers 15-40

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.

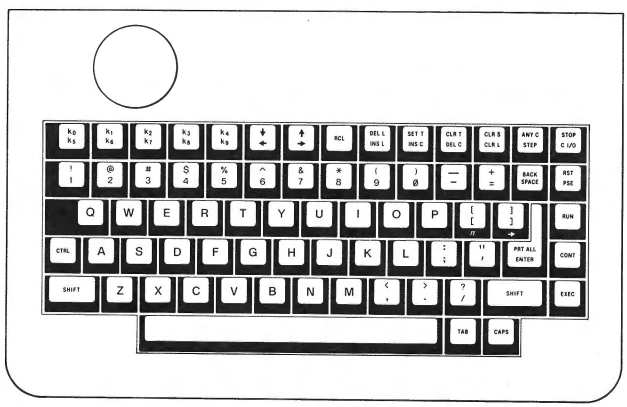


Figure 15-1. HP 98203A Keyboard

---

[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.

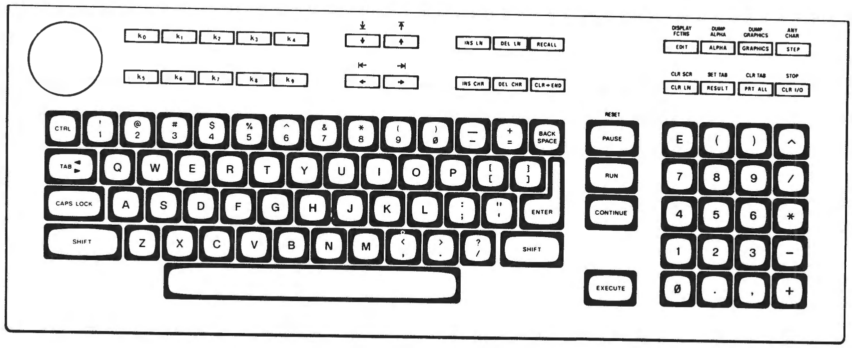


Figure 15-2. HP 98203B/C Keyboard

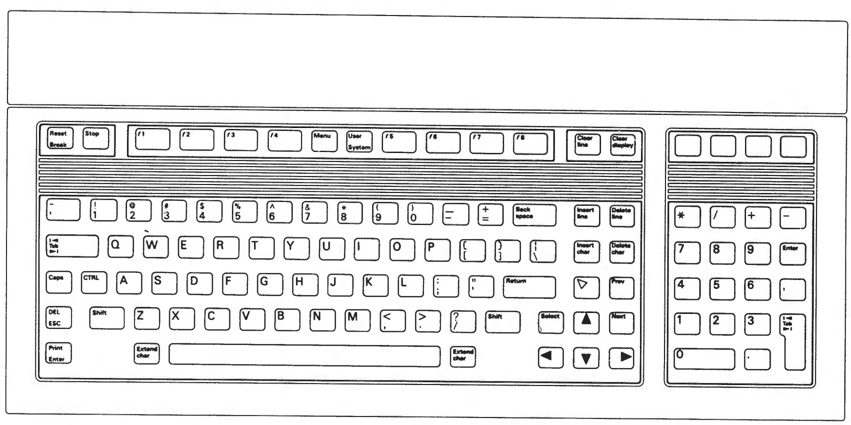


Figure 15-3. ITF Keyboard

Note that the ITF keyboard has only eight *physical* "user" function keys (f1 through f8, rather than k0 through k9), and lacks some of the *physical* "system" keys (such as ALPHA and RUN). However, ITF keyboards actually have *more* functionality than 98203 keyboards, because BASIC provides several "system" and "user" definitions for ITF function keys f1 through 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:

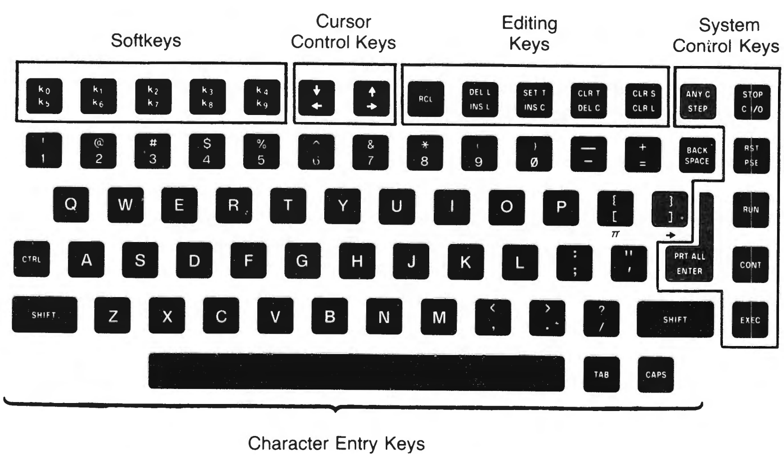


Figure 15-4. HP 98203A Keyboard

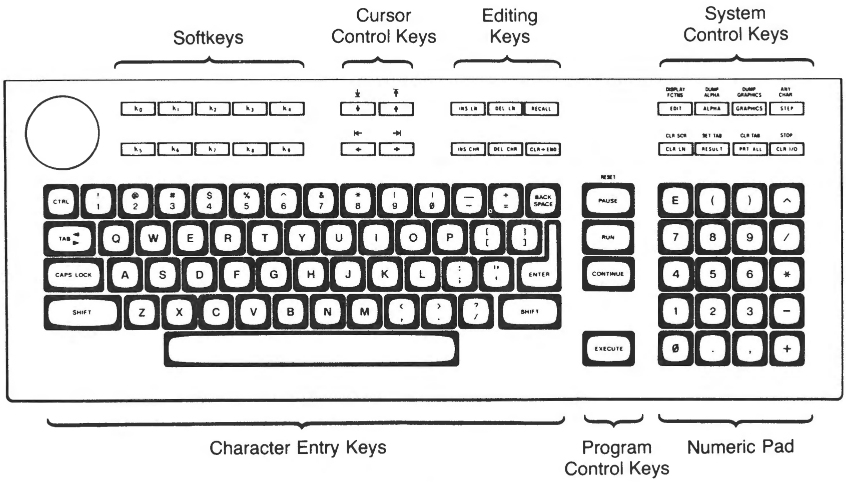


Figure 15-5. HP 98203B and C Keyboards

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.)

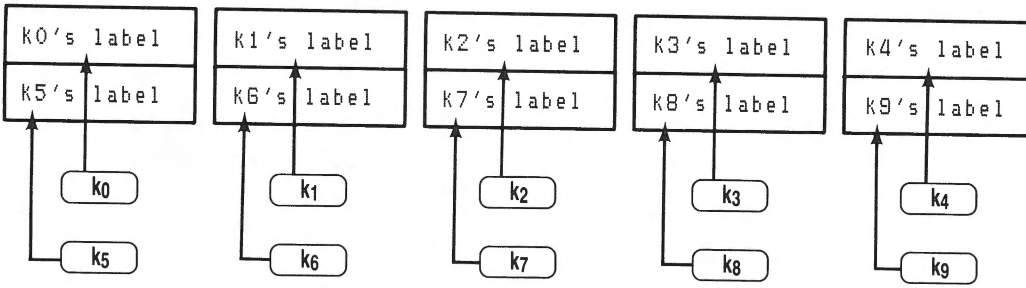


Figure 15-6. HP 98203 Softkey Labels

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:

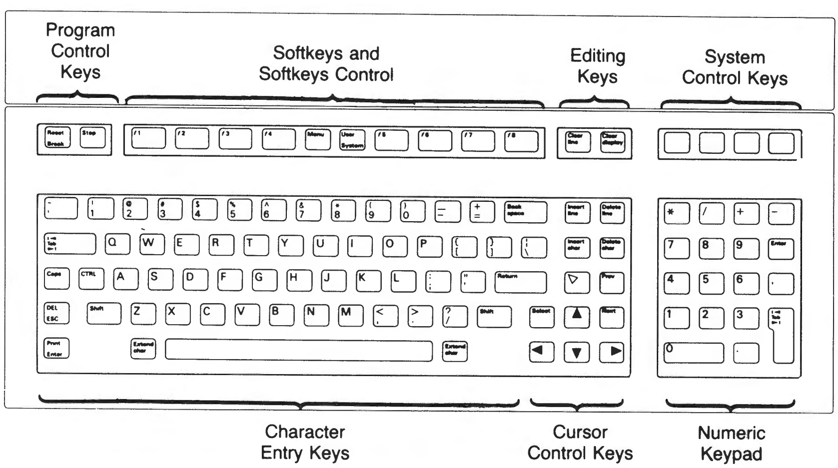


Figure 15-7. ITF Keyboards (such as the 46020)

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).

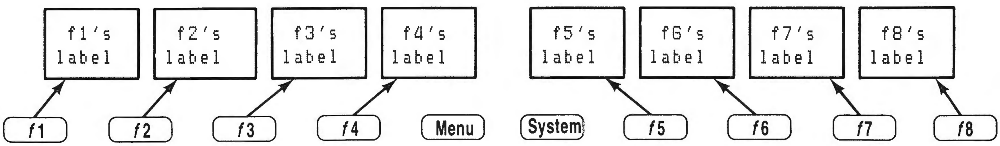


Figure 15-8. HIL "Typing-Aid" Softkey Labels

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:

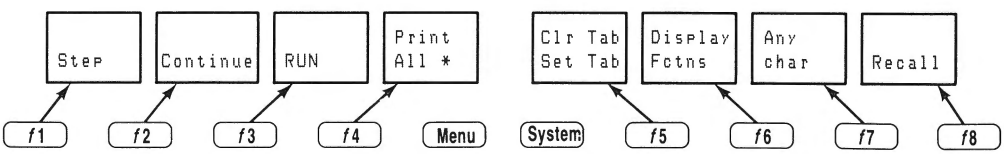


Figure 15-9. HIL "System" Menu Labels

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]:

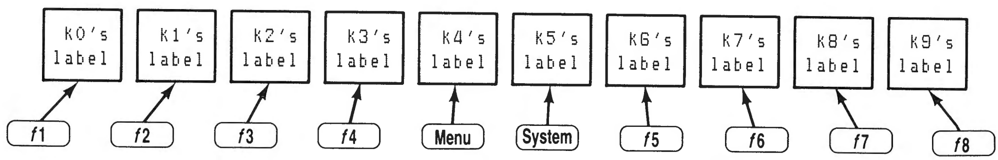


Figure 15-10. Correspondence Between Function Keys and Labels

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.

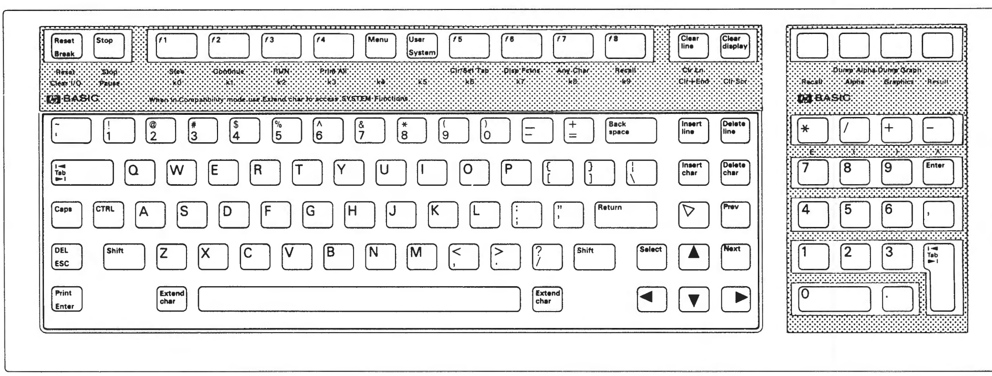
1 If you are in edit mode when you enter this compatibility mode, then edit mode is canceled.

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.

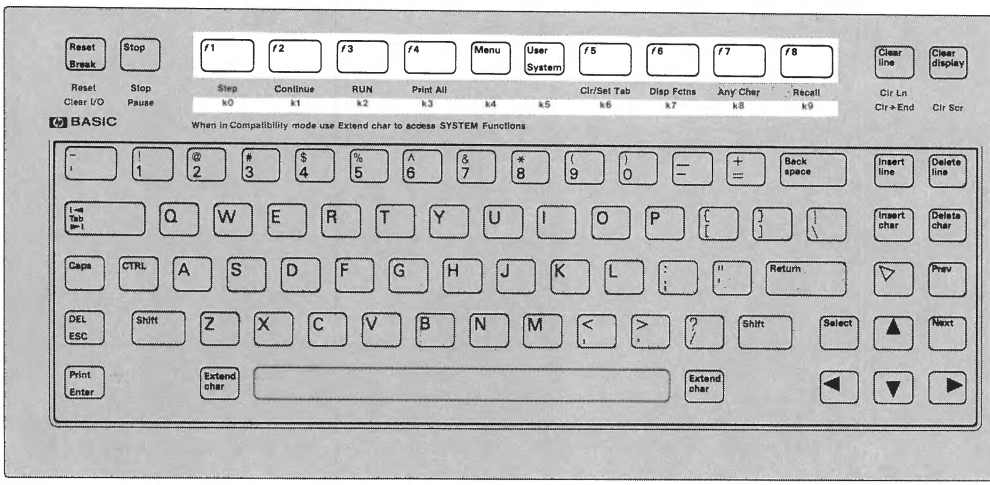
Note

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.

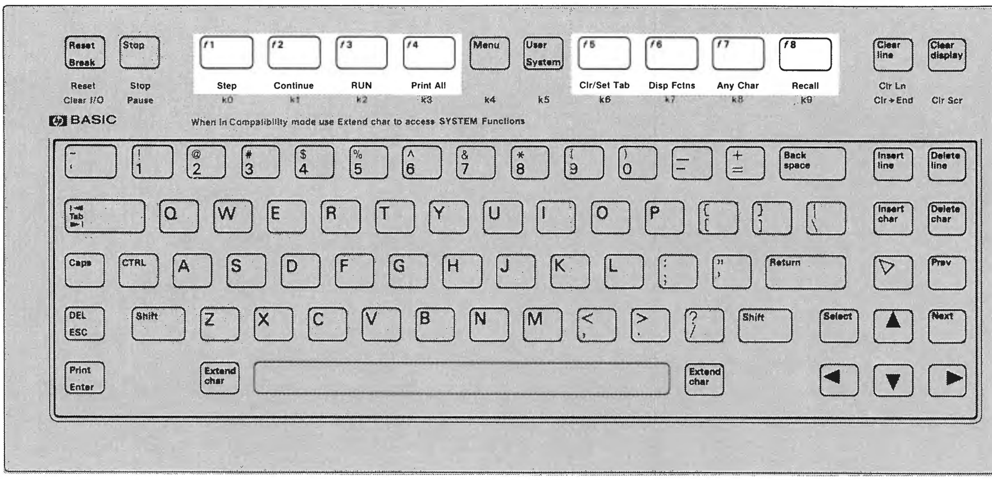


- To access a 98203 **softkey** definition, merely press the appropriate ITF softkey. For instance, the ITF [f1] softkey emulates the 98203 [k0] softkey, and the ITF [Menu] key emulates the 98203 [k4] softkey. (These key definitions are printed on the *bottom* row of the keyboard overlay.)

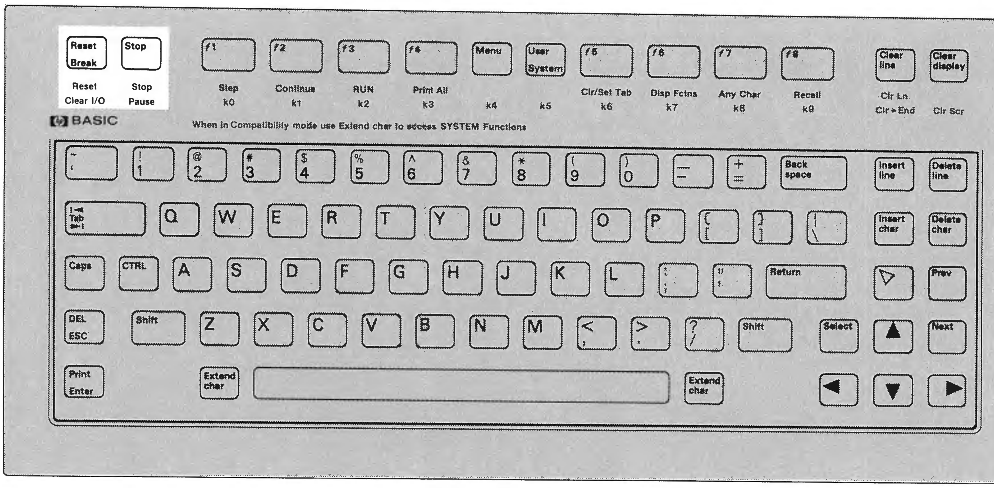
  Similarly, 98203 softkeys [k1] through [k1] are accessed by pressing the ITF [Shift] key with the appropriate softkey.



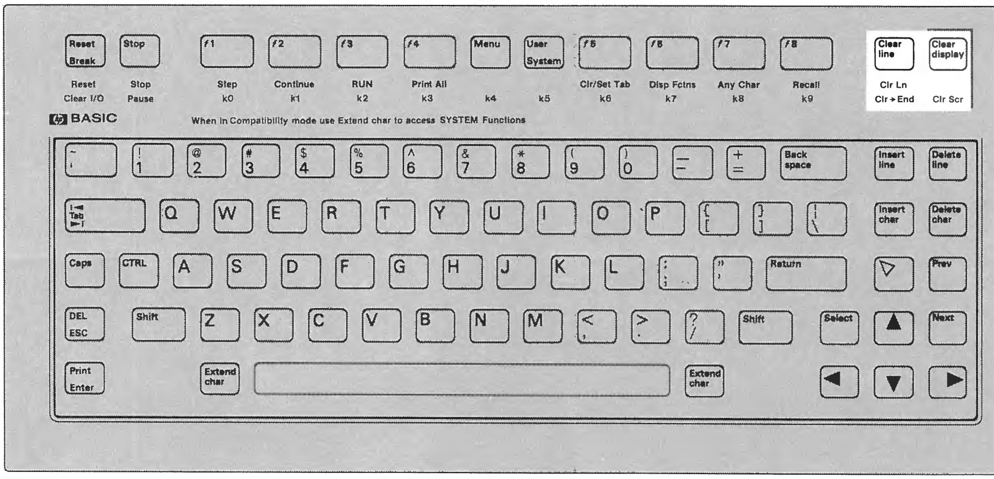
- To access a 98203 **system-key** definition, press [Extend char] with the appropriate ITF softkey. For instance, the ITF [Extend char]-[f1] key emulates the 98203 [STEP] key. (These key definitions are printed on the *top* row of the keyboard overlay. Note that these definitions are the same as in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



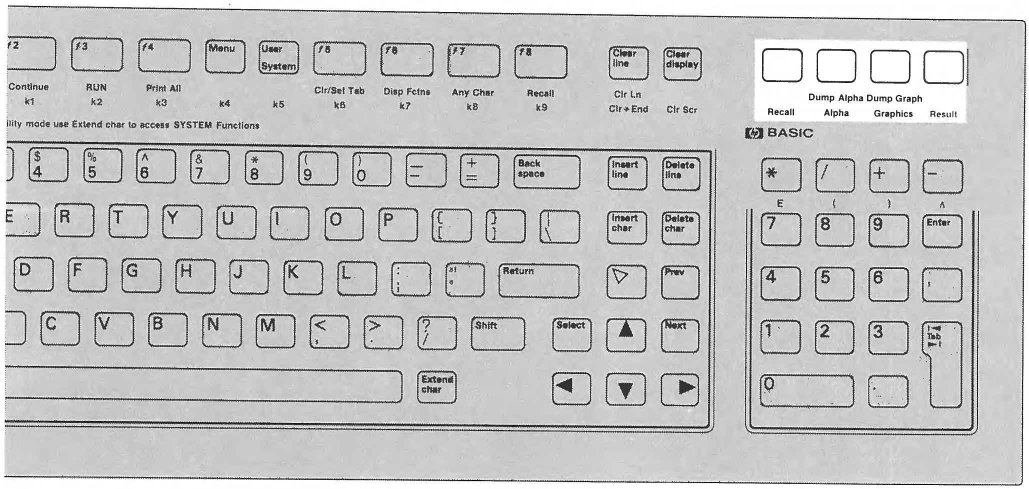
- The 98203 `CLR I/O` and `PAUSE` **system-key** definitions are available by using the ITF `Break` and `Stop` keys (*without* pressing `Extend char`). Note that these key definitions are the same in normal mode.



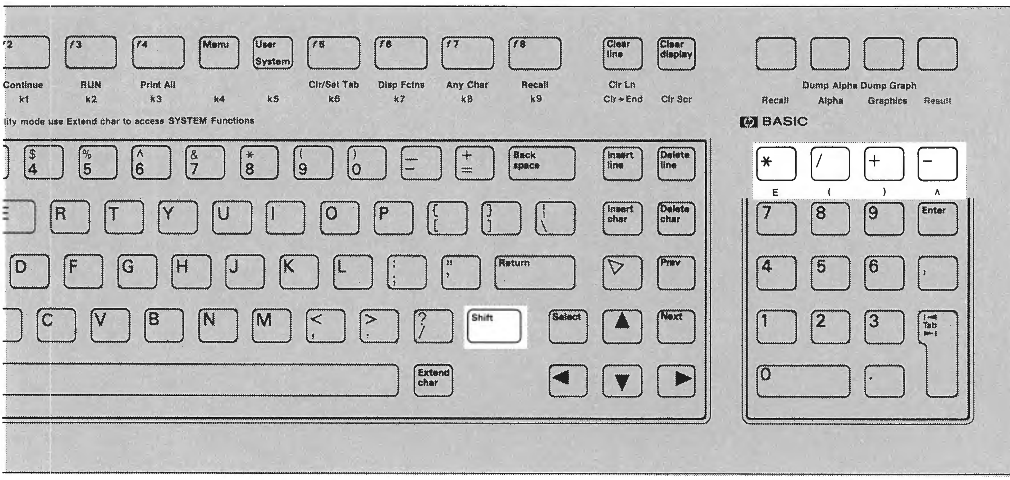
- The 98203 [CLR→END], [CLR LN], and [CLR SCR] **system-key** definitions are available by using the ITF [Clear line], [Shift] [Clear line], and [Clear display] keys. Note that these key definitions are the same in normal mode.



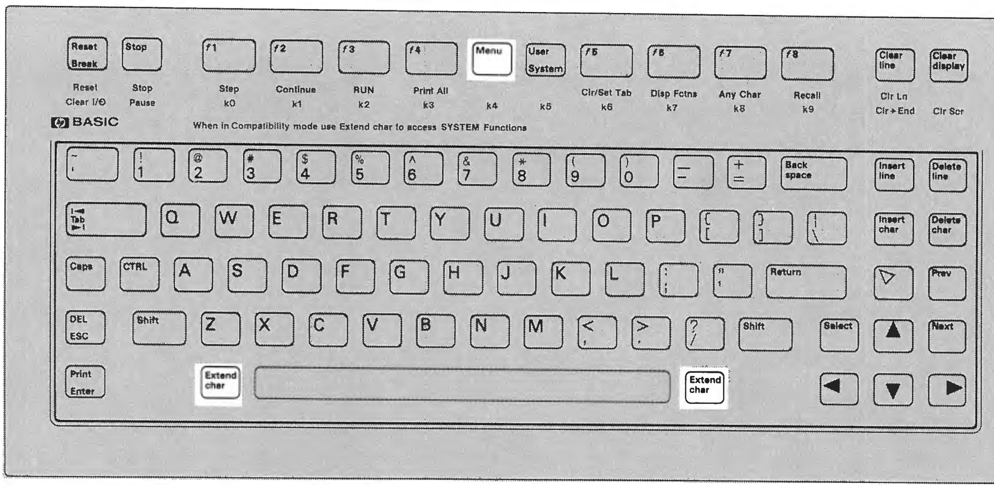
- The 98203 [RECALL], [ALPHA], [GRAPHICS], and **RES system-key** definitions are available by using the *unlabeled* ITF keys above the numeric keypad. The shifted keys also have corresponding definitions (for example, [Shift]-**Alpha** is the DUMP ALPHA function). Note that these key definitions are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



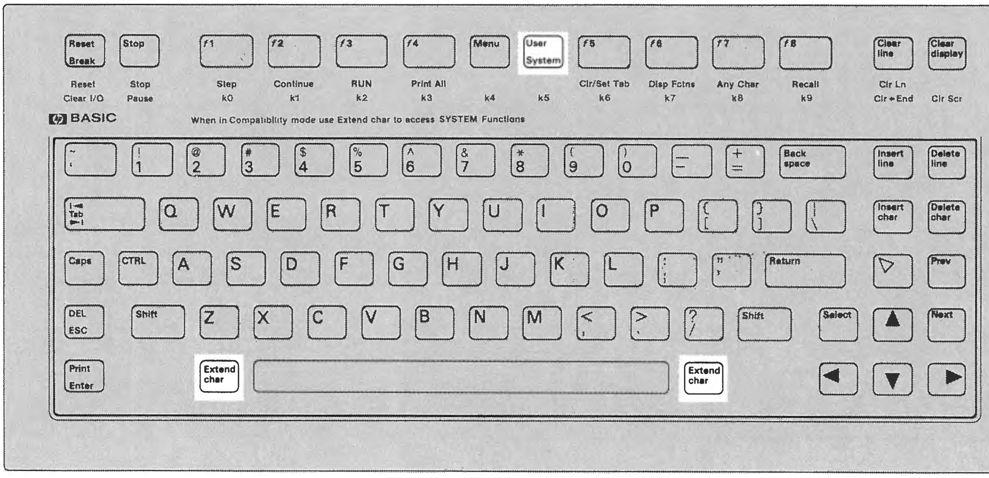
- When shifted, the [*], [/], [+], and [-] ITF keys on the top row of the numeric keypad have the same definitions as the keys on the top row of the 98203 numeric keypad. They are [E] ([Shift]-[*]), [(] ([Shift]-[/]), [)] ([Shift]-[+]), and [^] ([Shift]-[-]). Note that these key definitions are the same 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:

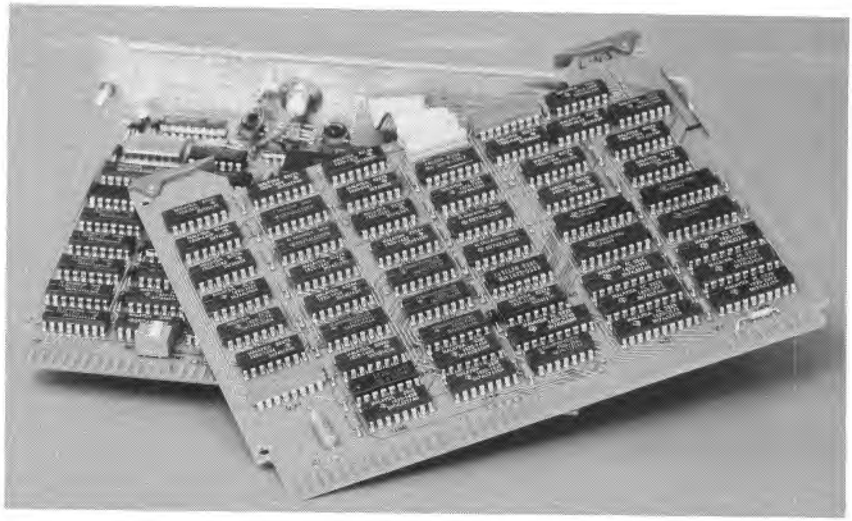


Figure 15-11. The HP 98546 Display Compatibility Interface

- 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.

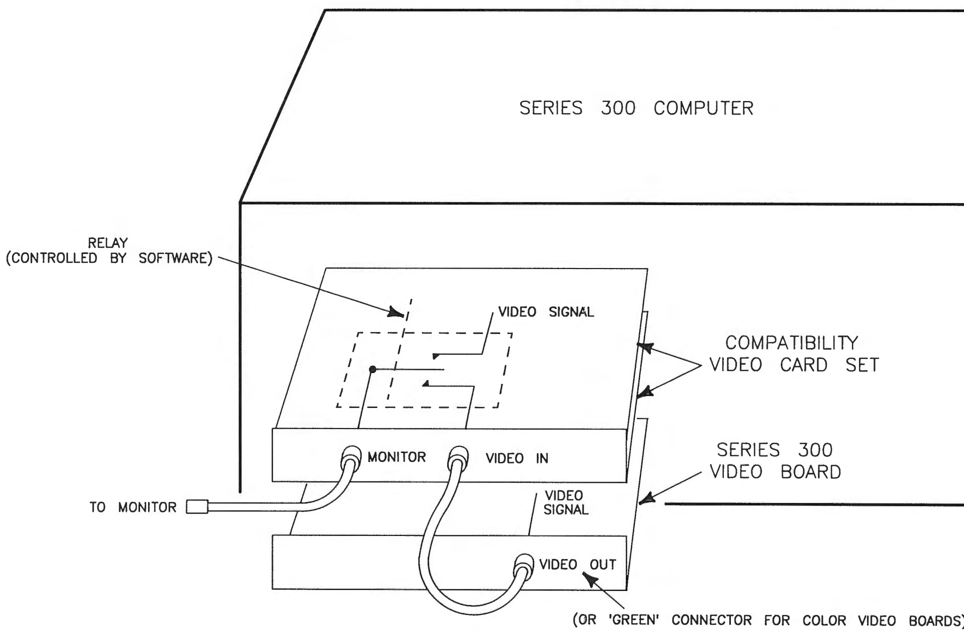


Figure 15-12. A Relay Governs Which Display Signal Is Used

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]: Tablets: HP 46087A (A size) HP 46088A (B size) 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: Model 236C Only.	CRT STATUS/CONTROL registers 5 (modified definition) and 15 through 17 (new). (See the *BASIC Language Reference.*)
New Graphics Utilities: 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: 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: 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]: Revised HIL "Swiss French*" and "Swiss German*" keyboards are now supported.	SYSTEM$("KEYBOARD LANGUAGE") returns corresponding identifier. (See the *BASIC Language Reference.*)
HPHIL Knob Interval Parameter[1]: None (same HIL knob as before).	With BASIC 3.0, the interval parameter for ON KNOB was ignored for HIL knobs. With 4.0, the parameter is used. (See the *BASIC Language Reference.*)
Read "Keyboard Input" Line (Non-Destructively): None.	Use SYSTEM$("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

Compatibility with Previous Versions 16-1
Categories of New Features 16-2
New Hardware Supported 16-2
New Utilities 16-3
HFS Disc Support 16-4
Human Interface Enhancements 16-5
New Keywords that Duplicate Register Operations 16-6
General Programming Additions 16-7
New STATUS/CONTROL Registers 16-8
Additional HP-HIL Support 16-9
Additional Graphics Features 16-10
Additional CSUB Capabilities 16-11

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` `DISPLAY FUNCTIONS OFF`	`CONTROL CRT,4;1` `CONTROL CRT,4;0`
`ALPHA PEN Pen_number`	`CONTROL CRT,5;Pen_number`
`KEY LABELS ON` `KEY LABELS OFF`	`CONTROL CRT,12;2` `CONTROL CRT,12;1`
`ALPHA HEIGHT Lines` `ALPHA HEIGHT`	`CONTROL CRT,13;Lines` 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` `KBD CMODE OFF`	`CONTROL KBD,15;1` `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: 0 ⇒ no battery-backed clock; 1 ⇒ HP 98270 battery-backed clock (Models 226 and 236 only); 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). 0 ⇒ scrolling enabled 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: 1 ⇒ enable (default) 0 ⇒ disable
CONTROL KBD,17;Disable_auto	Automatic menu switching: mode. <>0 ⇒ enable 0 ⇒ disable 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: • Most of the Pascal I/O Procedure Library • Some of the BASIC file I/O capabilities • Some display I/O capabilities • Some keyboard I/O capabilities • All SYSTEM$ capabilities	*CSUB Utility* manual

Porting and Sharing Files

17

Sharing HFS Discs and Data Files 17-2
General Compatibility Requirements 17-2
Common File Types 17-3
Common Data Types 17-4
HP-UX Text and Binary Files 17-5
Examples of HP-UX File Access: Textual Numeric Data 17-6
Porting LIF Files to SRM 17-24
SRM File Specifiers 17-24
SRM Mass Storage Volume Specification 17-25
Allowing for SRM Directory Paths 17-26
SRM Passwords vs. LIF Protect Codes 17-27
Copying Item-by-Item Using ENTER and OUTPUT 17-28
Accessing Files Created on Non-Series-200/300 SRM Workstations 17-29

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

Functionality Additions 18-1
Manual Changes 18-1

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*

The keyword LINK has also been added to the *BASIC Language Reference*. This keyword allows you to LINK a destination file on an HFS volume to a source file on the same volume. For example, executing the following statement:

```
LINK "Sor_file" TO "Des_file" [Return]
```

links the destination file `Des_file` to the source file `Sor_file`.

Index

a

ABS function ... 3-16, 4-68
Accent Priority ... 5-67
Access of Directories, Extended ... 7-53
Accessing Files Created on Non-Series-200/300 SRM Workstations ... 17-29
ACS function ... 3-17
ACSH function ... 3-18
Actual values ... 6-10
Adding Items to a Sorted List ... 5-25
Addresses, Primary ... 8-4
Allocate memory, Dynamically ... 3-3
ALLOCATE statement ... 3-3, 4-2, 4-7, 4-8, 4-25, 5-2, 12-3, 13-24
Alpha and Graphics Planes, Configuring Separate ... 15-29
Alpha Color Changes ... 15-38
Alpha Height, Changing ... 10-8
ALPHA HEIGHT statement ... 10-8
Alpha Screen Height and Graphics Scrolling ... 15-38
Alphanumeric Input, Accepting ... 10-36
Alternate CRT Characters ... 5-42
Anticipating Operator Errors ... 11-2
Appearance of Output ... 8-19
Arbitrary Exit Points ... 2-27
ARG function ... 4-68
Argument ... 3-25
Arithmetic Functions ... 3-15
Arithmetic Operations with Complex Arrays ... 4-28
Arithmetic Operators ... 3-9, 4-26
Array by Descending Subscripts, Searching an ... 4-40
Array, Copying a Subarray into an ... 4-20
Array, Dynamically Allocated ... 4-5
Array Element, Assigning an Individual ... 4-10
Array Fetches vs. Simple Variables ... 13-19
Array, four-dimensional ... 4-7
Array Functions ... 3-16
Array in Common ... 4-6

Array Indexing 13-12
Array Initialization 13-4
Array into a Subarray, Copying an 4-19
Array into Itself, Copying a Portion of an 4-22
Array Operations, Examples of Complex 4-67
Array, Planes of a Three-Dimensional REAL 4-3
Array, Printing an Entire 4-14
Array, Reordering an 5-28
Array, Searching a Three-Dimensional 4-43
Array, Summing the Elements in an 4-29
Array the Same Value, Assigning Every Element in an 4-11
Array, Three-Dimensional INTEGER 4-6
Array, Two-Dimensional COMPLEX 4-5
Array, Two-Dimensional REAL Array 4-4
Array, Using the READ Statement to Fill an Entire 4-11
Arrays and Arithmetic Operators 4-26
Arrays, Boolean 4-30
Arrays, Copying Entire Arrays into Other 4-12
Arrays, Extracting Single Values From 4-10
Arrays, Filling 4-11
Arrays for Code Conversion, Using 4-70
Arrays for Display, Examples of Formatting 4-14
Arrays of Sound Instructions 10-20
Arrays, Passing Entire 4-16
Arrays, Printing 4-14
Arrays, Redimensioning 4-24
Arrays, Reordering 4-32
Arrays, Searching Numeric 4-38
Arrays, Searching String 5-32
Arrays, Some Examples of 4-3
Arrays, Sorting 4-34
Arrays, Storage and Retrieval of 7-5
Arrays, String 5-3
ASCII and Custom Data Representations 7-30
ASCII Character Set, The 5-37
ASCII file 7-14, 7-17, 7-18, 7-23
ASCII File Access 17-18
ASCII File I/O, Example of 7-16
ASCII file I/O techniques 7-16
ASCII file type 7-19
ASCII files 7-9, 13-6

ASCII Files, A Closer Look at Using 7-16
ASCII Files, Data Representations in 7-17
ASCII Files, Formatted ENTER with 7-24
ASCII Files, Formatted OUTPUT with 7-19
ASCII format 7-27
ASCII Lexical Order 5-46
ASN function 3-17
ASNH function 3-18
ASSIGN statement 7-11, 7-12, 7-13, 7-14, 12-3, 17-6
Assignable priorities 2-34
Assigning an Individual Array Element 4-10
Assigning Every Element in an Array the Same Value 4-11
Assigning Variables 3-4
Assignment Surprise, Delayed 3-12
ATN function 3-17
ATNH function 3-18
Attributes, Assigning 7-13
Automaitc REORDER, Sorting with 4-34
Automatic Display Selection at System Boot 15-35
Automatic redimensioning 4-13
AVAILABLE ENTRIES table 7-54

b

Base Conversion Functions 3-30
BASE function 3-16, 4-9
BASIC 4.0 Enhancementsfor Series 200 Computers 15-42
BASIC and the Control Dial Box 10-34
BASIC Programs, Trapping Errors with 11-5
Battery-Backed Real-Time Clock 9-3, 15-6
BDAT and HP-UX Files, A Closer Look at 7-26
BDAT and HP-UX Files, Reading Data From 7-46
BDAT file 7-10, 7-11, 7-16, 7-23, 7-27, 7-28, 7-30, 7-33
BDAT File System Sector 7-31
BDAT files 7-9, 7-14, 7-19, 13-6
BDAT Internal Representations (FORMAT OFF) 7-27
BEEP statement 10-16
Benchmarking Techniques 13-7
BIN Files, Missing Driver 14-5
BINAND function 3-19
Binary Files, HP-UX Text and 17-5
Binary Functions 3-18

Binary Integers (HP-UX File Access) 17-14
Binary Real Values (HP-UX File Access) 17-12
Binary Strings (HP-UX File Access) 17-16
Binary tree 6-35, 6-36
BINCMP function 3-19
BINEOR function 3-19
BINIOR function 3-19
BIT function 3-19
Bits, Bytes, and Mode Types 5-61
Blank Lines, Printing 10-6
BNC Video Connectors, The Relay and 15-32
Boolean Arrays 4-30
Boundaries, keywords that define 2-12
Boundary Conditions 11-2
Branch Type, Choosing a 11-5
Branching on Clock Events 9-14
Branching Restrictions 9-18
Built-In Interfaces 15-7

C

Cache Memory, Enabling and Disabling 13-23
Cache Memory, MC68020 Internal 13-23
CALL 2-13
CALL statement 2-6, 6-4, 6-5, 6-19, 6-20, 6-22, 6-28
Calling Subprograms 12-4
Calling Subprograms from the Keyboard 6-22
Case Conversion 5-19
Case Conversions 5-46, 5-48, 5-50, 5-52, 5-54
CASE ELSE statement 2-12, 2-18
Case of I/O Transfers, Special 14-26
CASE statement 2-12, 2-19, 5-36
CAT statement 7-53
Catalog Header, Suppressing the 7-58
Cataloging Individual PROG Files 7-53
Cataloging Selected Files 7-58
Cataloging to a String Array 7-55
Categories of New Features (BASIC 5.0) 16-2
CAUSE ERROR statement 11-14
CDIAL statement 10-33
Cell Size, Determining Character 10-11
Cells, Character 10-9

Chaining Programs 2-39
Changes, Statement 14-6
Changing Alpha Height 10-8
Chapter Preview 1-2
Character Cell Size, Determining 10-11
Character Cells 10-9
Character, Example of Changing One 10-13
Character Font Storage in Memory 10-11
Character Replacement, "1 for 2" 5-64
Character Replacement, "2 for 1" 5-66
Character Set, CRT 5-16
Character Set, The Extended 5-41
Characters, Control 8-7
Characters, "Don't Care" 5-63
Characters, Finding "Missing" 5-42
Characters, Highlight 5-41
CHR$ function 2-21
CHR$ string function 5-15, 5-44
CHRX function 10-11
CHRY function 10-11
CLEAR ERROR statement 11-16
CLEAR SCREEN statement 10-49
Clearing Error Conditions 11-16
Clearing Graphics Rasters, Disabling and 10-6
Clearing the Screen 10-5
CLOCK binary 9-1
Clock Events, Branching on 9-14
Clock Functions and Example Programs, Using 9-11
Clock Range and Accuracy 9-2
Clock, Reading the 9-3
Clock, setting 3-29
Clock, Setting the 9-4
Clock Time Format 9-4
Clock Value, Initial 9-1
CLR I/O (Break) Key, The 12-16
[CLR I/O] key 12-16
CMPLX function 3-23, 4-69
Code Conversion, Using Arrays for 4-70
Color Changes, Hidden 15-40
Column vector 4-49
Columns of a Matrix, Summing Rows and 4-65

COM Blocks 6-15, 6-16
COM blocks 6-16, 6-17, 6-18, 6-26
COM Blocks, Hints for Using 6-17
COM statement 2-12, 2-39, 2-41, 2-42, 2-43, 4-2, 4-8, 5-2, 7-13
COM vs. Pass Parameters 6-16
Comments and Multi-character Identifiers 13-4
Common Data Types 17-4
Common File Types 17-3
Communicating with the Operator 10-1
Communication, Program/Subprogram 6-8
Comparisons Between Two REAL or COMPLEX Values 3-14
Comparisons, REAL and COMPLEX Numbers and 11-3
Comparisons, Rounding Errors Resulting from 3-20
Compatibility, GLOAD/GSTORE 15-39
Compatibility Interface Capabilities, Display 15-32
Compatibility Interface, Using the Display 15-30
Compatibility Mode 15-20
Compatibility Mode, Enabling Keyboard (KBD CMODE ON) 15-19
Compatibility Mode, Exiting Keyboard (KBD CMODE OFF) 15-28
Compatibility Mode, HP 98203 Keyboard 15-14
Compatibility with Preceding Versions 14-3
Compatibility with Previous Versions (BASIC 5.0) 16-1
COMPLEX 3-1
COMPLEX Arguments and the Trigonometric Mode 3-24
Complex Array Operations, Examples of 4-67
Complex Arrays, Performing Arithmetic Operations with 4-28
COMPLEX Data Type 3-2
Complex Functions 3-22
COMPLEX Numbers, An Application for 3-26
COMPLEX Numbers and Comparisons, REAL and 11-3
COMPLEX Numbers, Determining the Parts of 3-24
COMPLEX Numbers, Evaluating 3-23
COMPLEX statement 4-2, 4-8
COMPLEX value 4-1
COMPLEX Values, Creating 3-23
COMPLEX variables 3-4
Composition of SRM File Names 17-24
Computer's Resources, Efficient Use of the 13-1
Concatenation, String 5-4
Concatenation vs. Substring Placement 13-19
Conditional Branching 2-13

Conditional execution 2-10
Conditional Execution of One Segment 2-11
Conditional GOTO expressed 2-13
Conditional GOTO implied 2-13
Conditional segment 2-11
Conditional Segments, Multiple-Line 2-14
Conditional Subroutine 2-13
Configuration Program, Using a 15-12
Configurations Possible 15-33
Configuring, and Verifying Your Printer, Installing, ... 8-2
Configuring BASIC 14-3
Configuring Separate Alpha and Graphics Planes 15-29
CONJG function 3-24, 4-69
Constants 13-14
CONT statement 2-4
Context Switching 6-19
CONTINUE key 2-4, 2-5
Continuing a Program, Pausing and 12-5
Control Characters 8-7
Control Characters, Displaying 5-38
Control Dial Box and BASIC 10-34
Control Dial Handler, An Example 10-35
Control Dials, Using 10-33
Conversion, Case 5-19
Conversion, Number-Base 5-34
Conversions, Case 5-46, 5-48, 5-50, 5-52, 5-54
Conversions, Implicit Type 3-4
Conversions, Type 13-13
Converting from Rectangular to Polar Coordinates 3-25
Copying a Portion of an Array into Itself 4-22
Copying a Subarray into an Array 4-20
Copying a Subarray into another Subarray 4-21
Copying an Array into a Subarray 4-19
Copying Entire Arrays into Other Arrays 4-12
Copying Item-by-Item Using ENTER and OUTPUT 17-28
Copying Subarrays 4-16
Copying Subarrays, Rules for 4-23
COS function 3-17
COSH function 3-18
CREATE BDAT statement 7-32
CREATE statement 7-16, 17-5

Creating COMPLEX Values 3-23
Cross Reference, Example Program and 12-7
Cross References 12-6
Cross-Reference Listing, Generating a 12-6
CRT Character Set 5-16
CRT Characters, Alternate 5-42
CRT function 3-31
CRTX binary 10-5
CSUB Capabilities (BASIC 5.0), Additional 16-11
CSUB Utility 18-1
CSUBs 14-6
CSUBs, Incompatible 15-37
Current Height 10-7
Cursor-control routine 2-35
Custom Character Fonts 10-9
Custom Keyboard Interface, An Example 10-55
Cycles and Delays 9-15

d

DATA and READ Statements, Using 7-3
Data Files 13-6
Data From a File, Reading String 7-46
Data From BDAT and HP-UX Files, Reading 7-46
Data in Programs, Storing 7-2
Data in Variables, Storing 7-2
Data Input by the User 7-2
Data Pointer, Moving the 7-6
Data Representations, ASCII and Custom 7-30
Data Representations Available 7-26
Data Representations in ASCII Files 7-17
Data Representations with HP-UX Files 7-30
DATA statement 2-12, 4-12, 6-19, 7-1, 7-3, 7-4
Data Storage 13-1
Data Storage and Retrieval 7-1
Data Storage in Read/Write Memory 13-1
Data Stroage on Mass Memory Devices 13-3
Data Structure 6-34
Data, Textual Numeric 17-6
Data Type, COMPLEX 3-2
Data Type, INTEGER 3-2
Data Type, REAL 3-1

Data Type Stroage Requirements 13-3
Data Types, Common 17-4
Data, Writing 7-37
Data-Type Matching Between BASIC and C 17-5
Date and Time of Day, Determining the 9-3
Date format, European 9-10
DATE function 3-29
Date Functions, Time and 3-29
Date, Setting Only the 9-8
DATE$ string function 9-3
Dates, Days Between Two 9-13
Day of the Week 9-13
Day, Time of 9-16
Days Between Two Dates 9-13
DCOMM binary 9-1
Deactivated interrupt 2-36
Deactivating events 2-36
DEALLOCATE statement 3-3, 13-24
Debugging Programs 12-1
Declaration of variables, keywords used in the 2-12
Declaring Variables 3-3
DEF FN statement 2-12, 6-8, 6-28, 6-29, 6-30
Default dimensioned length of a string 5-1
Default mass storage device 7-10
Default Plotter 14-14
Default range 4-18
Default Soft Font, Restoring the 10-13
Defined Records 7-31
Defining Typing-Aid Softkeys Programmatically 10-25
DEG statement 3-17, 3-24, 6-20
Degradation, rate 7-19
Degrees 3-17
DEL LN statement 6-29
Delayed Assignment Surprise 3-12
Delays, Cycles and 9-15
Deleting Subprograms 6-27, 6-29
DELSUB statement 6-27
DES a secondary word 4-35
DES secondary keyword 4-40
Description of Series 300 Hardware 15-3
Design, Top-Down 6-33

DET function 3-16, 4-59
Detecting Ill-conditioned Matrices 4-61
Determinant 4-59
Determinant of a Matrix, The 4-59
Determining Character Cell Size 10-11
Determining Error Number and Location 11-7
Device Selectors 8-3
Device Selectors, Using 8-6
Device Viewport, Input 14-14
Dials, Using Control 10-33
DIM statement 2-12, 3-3, 4-2, 5-2
Dimension table 13-1
dimension table 13-1
Dimensioning, Problems with Implicit 4-8
Directories, Extended Access of 7-53
Directory Paths, Allowing for SRM 17-26
DISABLE statement 2-38, 6-20, 6-21, 11-6
Disabled interrupt 2-36
Disabling and Clearing Graphics Rasters 10-6
Disabling and Enabling Alpha Scrolling 10-4
Disabling Display Functions Mode 10-4
Disabling Error Trapping (OFF ERROR) 11-6
Disabling Events 2-38
Disabling Printall Mode 10-4
Disjoint binary trees 6-35
Display Compatibility Interface Capabilities 15-32
Display Compatibility Interface, Using the 15-30
Display Drivers, Removing 15-35
Display Functions 14-25
Display Functions Mode, Disabling 10-4
Display Selection at System Boot, Automatic 15-35
Displaying and Prompting 10-3
Displaying Characters on the Screen 10-8
Displaying Messages 10-3
Displays 15-4
Documentation 14-3
Don't Care Characters 5-63
DOT function 3-16, 4-51
Double-Subscript Substrings 5-7
Driver BIN Files, Missing 14-5
DROUND function 3-20, 3-21, 11-4

DUMP DEVICE IS statement 8-17
DVAL function 3-30, 5-34
DVAL$ string function 5-34
Dyadic operator 3-12
Dynamically allocate memory 3-3
Dynamically Allocated, Two-Dimensional INTEGER Array 4-5

e

Editing Subprograms 6-28
Editor, A Simple Music 10-18
Editor, Font 10-14
Editor Utility Capabilities, Font 10-14
Efficient Use of the Computer's Resources 13-1
Elements in an Array, Summing the 4-29
ENABLE statement 2-38
Enabling Alpha Scrolling, Disabling and 10-4
Enabling and Disabling Cache Memory 13-23
Enabling and Disabling Floating-Point Math Hardware 13-21
Enabling Keyboard Compatibility Mode 15-19
END IF statement 2-12
END LOOP statement 2-12
END SELECT statement 2-12, 2-18
END statement 2-3, 2-12, 6-5
END WHILE statement 2-12
End-Of-File pointers 7-35
End-of-line (EOL) sequences 7-13
End-Of-Record 7-33
End-Of-Record (EOR) 7-43
Enhancements, 5.1 18-1
ENTER, Random 7-48
ENTER, Serial 7-47
ENTER statement 7-11, 7-19, 7-24, 7-46
Entering a Single Item 10-38
EOF and EOR Conditions, Trapping 7-50
EOF Pointer 7-36
EOF pointer 7-13
EOF pointer, Logical 7-40
EOF pointer, Physical 7-40
EOF Pointers 7-35
EOF pointers 7-36
EOF Pointers, Moving 7-36

Equations, Solving Simultaneous 4-55
ERRDS function 11-8
ERRL function 11-7, 11-14
ERRL in Subprograms, Using ERRLN and 11-12
ERRLN and ERRL in Subprograms, Using 11-12
ERRLN function 11-7, 11-14
ERRM$ string function 11-8, 11-14
ERRN function 11-7, 11-14
Error Conditions, Clearing 11-16
Error, Example of Simulating an 11-15
Error Number and Location, Determining 11-7
Error Responses, Overview of 11-1
ERROR RETURN statement 11-8
Error Trapping and Recovery, Scope of 11-6
Error Trapping (OFF ERROR), Disabling 11-6
Errors, Anticipating Operator 11-2
Errors, Handling 11-1
Errors with BASIC Programs, Trapping 11-5
Escape-Code Sequences 8-8
European date format 9-10
Evaluating COMPLEX Numbers 3-23
Evaluating Expressions Containing Strings 5-4
Evaluating Scalar Expressions 3-9
Evaluation Hierarchy 5-4
Evaluations, Polynomial 13-14
Event-checking 2-30
Event-initiated branching 2-2, 2-30, 2-32
Event-initiated RECOVER statement 6-21
Events, Branching on Clock 9-14
Events, Disabling 2-38
Events, Types of 2-31
Executing Commands While a Program Is Running 12-2
Executing Example SOUND Instructions 10-21
EXIT IF statement 2-12, 2-29
Exiting Keyboard Compatibility Mode 15-28
EXP function 3-17
Expanded Softkey Menu, An 10-45
Exponential Functions 3-17
Exponentiation vs. Multiply and SQRT 13-18
Expressions as Pass Parameters 3-13
Expressions, hierarchy for 3-9

Extended Access of Directories 7-53
Extended Character Set, The 5-41
External Printer, Using the 8-7

f

File Access, A Closer Look at General 7-11
File Access, ASCII 17-18
File Dump Utility, HP-UX 17-22
File Input and Output 7-7
File pointer 7-23
File specifier 7-10
File Specifiers, SRM 17-24
File Types, Brief Comparison of Available 7-7
File Types, Common 17-3
Files Cataloged, Getting a Count of 7-57
Files, Cataloging Selected 7-58
Files Created on Non-Series-200/300 SRM Workstations, Accessing 17-29
Files, Data 13-6
Files, Getting a Count of Selected 7-60
Files, Skipping Selected 7-61
Files, Storing and Loading Typing-Aids from 10-25
Files to the Spooler Directories, Writing 8-18
FIND statement 12-6
Floating-Point Math Card, HP 98635 13-21
Floating-Point Math Co-Processor, MC68881 13-21
Floating-Point Math Hardware, Enabling and Disabling 13-21
FN statement 2-6
FNEND statement 2-12, 6-27, 6-30
Folating-point math card 6-22
Font Editor 10-14
Font Editor Utility 10-15
Font Editor Utility Capabilities 10-14
Font, Re-Defining an Entire 10-15
Font Storage in Memory, Character 10-11
Font Usage, Soft 10-12
Fonts, Custom Character 10-9
FOR statement 2-12, 2-24
Formal parameter list 6-11
Formal Parameter Lists 6-9
FORMAT attribute 7-14
Format, Clock Time 9-4

FORMAT OFF statement 7-13, 7-14, 7-27
FORMAT ON attribute 7-19
FORMAT ON statement 7-13, 7-24, 7-27
Formatted ENTER with ASCII Files 7-24
Formatted OUTPUT with ASCII Files 7-19
Formatted Printing 8-9
Formatting Arrays for Display 4-14
FOR...NEXT structure 2-22, 2-25, 2-26, 2-27
Four-dimensional array 4-7
FRACT function 3-16
FRENCH Lexical Order 5-48
Function, ABS 3-16, 4-68
Function, ACS 3-17
Function, ACSH 3-18
Function and a Subprogram, Difference 6-6
Function, ARG 4-68
Function, ASN 3-17
Function, ASNH 3-18
Function, ATN 3-17
Function, ATNH 3-18
Function, BASE 3-16
Function, BINAND 3-19
Function, BINCMP 3-19
Function, BINEOR 3-19
Function, BINIOR 3-19
Function, BIT 3-19
Function, CHRX 10-11
Function, CHRY 10-11
Function, CMPLX 3-23, 4-69
Function, CONJG 3-24, 4-69
Function, COS 3-17
Function, COSH 3-18
Function, CRT 3-31
Function, DATE 3-29
Function, DET 3-16, 4-59
Function, DOT 3-16, 4-51
Function, DROUND 3-20, 3-21, 11-4
Function, DVAL 3-30, 5-34
Function, ERRDS 11-8
Function, ERRL 11-7, 11-14
Function, ERRLN 11-7, 11-14

Function, ERRN 11-7, 11-14
Function, EXP 3-17
Function, FRACT 3-16
Function, IDN 4-52
Function, IMAG 3-24, 4-67
Function, INT 3-16
Function, INV 4-53
Function, IVAL 3-30, 5-34
Function, KBD 3-31
Function, KNOBX 14-9
Function, LGT 3-17
Function, LOG 3-17
Function, MAX 3-19, 5-33
Function, MAXREAL 3-16
Function, MIN 3-19, 5-33
Function, MINREAL 3-16
Function, NUM 5-14
Function, PI 3-17
Function, POS 5-12, 5-15
Function, PROUND 3-20
Function, PRT 3-31
Function, RANK 3-16
Function, REAL 3-24, 4-67
Function, RES 3-31
Function, RND 3-22
Function, ROTATE 3-19
Function, SC 3-31
Function, SGN 3-16
Function, SHIFT 3-19
Function, SIN 3-17
Function, SINH 3-18
Function, SIZE 3-16
Function, SQR 3-16
Function, SQRT 3-16, 3-23
Function, SUM 3-16
Function, TAB 10-6
Function, TAN 3-17
Function, TANH 3-18
Function, TIME 3-29, 9-6
Function, TIMEDATE 3-29, 9-3, 9-6
Function, Transpose 4-63

Function, TRN 4-63
Function, VAL 5-13
Function, VAL$ 7-23
Functions and String Functions, REAL Precision 6-6
Functions, Arithmetic 3-15
Functions, Array 3-16
Functions, Base Conversion 3-30
Functions, Binary 3-18
Functions, Complex 3-22
Functions, Exponential 3-17
Functions, General 3-31
Functions, Hyperbolic 3-18
Functions, Limit 3-19
Functions, Numerical 3-15
Functions, Rounding 3-19
Functions, Step 3-13
Functions, String 5-17
Functions, String-Related 5-10
Functions, Subprograms and User-Defined 6-1
Functions, Time and Date 3-29
Functions, Trigonometric 3-17

g

GCLEAR, Implicit 14-14
GCLEAR statement 10-6
General File Access, A Closer Look at 7-11
General Functions 3-31
Generating a Cross-Reference Listing 12-6
Generating Sound 10-16
GERMAN Lexical Order 5-50
GET statement 2-39, 2-40, 2-41, 6-17, 13-5
GET, Using 2-40
GLOAD/GSTORE Compatibility 15-39
GOSUB statement 2-6, 2-7, 2-8, 6-19, 6-20
GOTO statement 2-6, 2-7, 2-13, 2-27, 6-19, 6-20
Graphics Features (BASIC 5.0), Additional 16-10
GRAPHICS INPUT IS statement 10-32
GRAPHICS OFF statement 10-6

Graphics Planes, Configuring Separate Alpha and 15-29
Graphics Rasters, Disabling and Clearing 10-6
Graphics Scrolling, Alpha Screen Height and 15-38
Graphics Tablet DIGITIZE 14-15

h

Halting Program Execution 2-3
Handling Errors 11-1
Hardware Description 15-31
Hardware, Description of Series 300 Hardware 15-3
Hardware Supported (BASIC 5.0), New 16-2
Height, Changing Alpha 10-8
Height, current 10-7
Height, Determining Screen Width and 10-7
Helpful Documentation 14-3
HFS Disc Support 16-4
HFS Discs and Data Files, Sharing 17-2
HFS "Extended" Catalog, Getting an 7-57
Hidden Color Changes 15-40
Hierarchical File System (HFS) 7-9, 16-4
Hierarchy, Evaluation 5-4
Hierarchy for expressions 3-9
Hierarchy, Math 3-10
Highlight Characters 5-41
HIL "System" Menu Labels 15-18
HIL "Typing-Aid" Softkey Labels 15-18
HP-HIL Keyboard Interface 15-8
HP-HIL Keyboards with Mouse 14-11
HP-HIL Knob Interval Parameter 15-41
HP-HIL Support (BASIC 5.0), Additional 16-9
HP-UX Binary Files 17-5
HP-UX File 7-14, 7-31
HP-UX File Dump Utility 17-22
HP-UX File Terminology, A Note About 17-2
HP-UX Files, Data Representations with 7-30
HP-UX Text and Binary Files 17-5
HP-UX Text Files 17-5, 17-7
HP 2225 ThinkJet(TM) Printer 8-1
HP 2563 Dot-Matrix Impact Printer 8-1
HP 2565 Dot-Matrix Impact Printer 8-1
HP 2566 Dot-Matrix Impact Printer 8-1

HP 2601 Daisy-Wheel Impact Printer 8-1
HP 2602 Daisy-Wheel Impact Printer 8-1
HP 2671 Thermal Printer 8-1
HP 2673 Thermal Printer 8-1
HP 2686 LaserJet(TM) Printer 8-1
HP 2932 Dot-Matrix Impact Printer 8-1
HP 2934 Dot-Matrix Impact Printer 8-1
HP 3630A (PaintJet(TM)) 18-1
HP 82906 Dot-Matrix Impact Printer 8-1
HP 98203 Keyboard Compatibility Mode 15-14
HP 98203 Softkey Labels 15-16
HP 98203 Specific Key Codes 15-37
HP 98548A 18-1
HP 98549A 18-1
HP 98550A 18-1
HP 98635 Floating-Point Math Card 13-21
HP 98644 Serial Interface Configuration 15-12
HP 98646A VME Interface 18-1
HP 9876 Thermal Printer 8-1
Human Interface Enhancements (BASIC 5.0) 16-5
Human Interfaces, Example 10-45
Human I/O Mechanisms, Overview of 10-2
Hyperbolic Functions 3-18

i

ID PROM 15-10
Identity Matrix 4-52
IDN function 4-52
IF statement 2-12
IF...THEN statement 2-11, 2-12, 2-13
IF...THEN structure 2-29
IF...THEN...ELSE statement 2-16
Ill-Conditioned Matrices 4-60
Ill-conditioned Matrices, Detecting 4-61
IMAG function 3-24, 4-67
Image Specifiers, Additional 8-15
Image Specifiers, Numeric 8-12
Image Specifiers, String 8-14
Images, Using 8-11
Implicit Dimensioning, Problems with 4-8
Implicit GCLEAR 14-14

Implicit Type Conversions 3-4
Incompatible CSUBs 15-37
Indexing, Array 13-12
Individual Array Elements, Using 4-10
Infinite loop 2-7, 2-32, 2-33, 2-34
Initial Clock Value 9-1
Initialization, Variable 6-20
Input, Accepting Alphanumeric 10-36
Input Device Viewport 14-14
Input, Operator 10-24
INPUT statement 5-19, 7-2, 10-36, 10-39, 12-10
Inputs, Softkey 10-24
Inserting Subprograms 6-28
Installing, Configuring, and Verifying Your Printer 8-2
Instructions, Arrays of Sound 10-20
INT function 3-16
INTEGER 3-1
INTEGER Data Type 3-2
INTEGER statement 2-12, 4-2, 4-8
INTEGER value 4-1
INTEGER Variables 13-9
Interface Capabilities, Display Compatibility 15-32
Interface Configuration, HP 98644 Serial 15-12
Interface Enhancements (BASIC 5.0), Human 16-5
Interface, HP-HIL Keyboard 15-8
Interface select code 8-3
Interface, Serial 15-7
Interfaces, Built-In 15-7
Internal Numeric Formats 3-8
Internal real-time clock 9-14
Interrupt, deactivated 2-36
Interrupt, disabled 2-36
Interval Timing 9-13
INV function 4-53
Inverse Matrix 4-53
I/O path 7-12
I/O path name 7-10, 7-12, 7-13
I/O Path, Opening an 7-12
I/O Paths, Closing 7-15
I/O techniques, ASCII file 7-16
I/O Transfers, Special Case of 14-26

ITF Keyboards (such as the 46020) 15-17
IVAL function 3-30, 5-34
IVAL$ string function 5-34

k

KBD function 3-31
Key Codes, HP 98203 Specific 15-37
KEY LABELS OFF statement 10-51
KEY LABELS ON statement 10-51
Key specifier 4-44
Keyboard, Calling Subprograms from the 6-22
Keyboard Commands Disallowed During Program Execution 12-5
Keyboard Compatibility Mode, Enabling 15-19
Keyboard Compatibility Mode, Exiting 15-28
Keyboard Compatibility Mode, HP 98203 15-14
Keyboard Interface, An Example Custom 10-55
Keyboard Interface, HP-HIL 15-8
Keyboard Layouts, Brief Comparison of 15-14
Keyboards with Built-in Knob 14-10
Keywords and Capabilities 10-33
Keywords Duplicating Register Operations 16-6
Keywords that define boundaries 2-12
Keywords that define program structures 2-12
Keywords that Duplicate Register Operations (BASIC 5.0), New 16-6
Keywords used in the declaration of variables 2-12
Keywords used to identify lines that are literals 2-12
KNB2_0 14-13
Knob 14-9
Knob, Example of Using 2-34
Knob Interval Parameter, HP-HIL 15-41
Knob, Keyboards with Built-in 14-10
Knobs, Using 10-32
KNOBX function 14-9

l

LABEL, keyword 2-33
LABEL with PIVOT 14-22
Labels, Softkey 10-28
Language Extensions BIN Files, Missing 14-4
Length header, string variable's 7-20

LET statement 3-4, 7-2
LEX binary 5-5
LEX_AID program 5-58
Lexical Order, ASCII 5-46
Lexical Order, FRENCH 5-48
Lexical Order, GERMAN 5-50
Lexical Order, Introduction to 5-36
LEXICAL ORDER IS 5-19
LEXICAL ORDER IS statement 5-5, 5-36, 5-37, 5-43, 5-58, 5-60, 5-62
Lexical Order, Predefined 5-43
Lexical Order, SPANISH 5-52
Lexical Order, SWEDISH 5-54
LEXICAL ORDER, User-defined 5-56
Lexical Orders, User-Defined 5-58
Lexical Tables 5-45
LGT function 3-17
Libraries, Using Subprograms 6-24
LIF file 7-12
LIF Files to SRM, Porting 17-24
LIF Protect Codes, SRM Passwords vs. 17-27
Limit Functions 3-19
Linear Flow 2-3
Linear flow 2-2
LINK statement 18-1
LINPUT statement 5-19, 7-2, 7-35, 10-36, 10-39, 12-10
LINPUT with Multiple Fields 10-41
LIST KEY statement 10-25
Listing Current Typing-Aid Softkey Definitions 10-25
Literals, keywords used to identify lines that are 2-12
Live Keyboard, Using 12-2
LOAD command 2-39, 2-40
LOAD KEY statement 10-25
LOAD statement 2-39, 6-17, 6-26, 13-5
Loading and Running Programs, Just 15-11
Loading Several Subprograms at Once 6-25
Loading Subprograms 6-24
Loading Subprograms One at a Time 6-25
Loading Subprograms Prior to Execution 6-26
Loading Typing-Aids from Files, Storing and 10-25
LOADSUB statement 6-25, 9-7
LOADSUB...FROM statement 6-24, 6-26, 6-27

LOC condition field 4-40, 4-42
LOC MAX condition field 4-39, 4-42
LOC MIN condition field 4-40, 4-42
LOG function 3-17
Logical Comparisons for Equality on REAL Numbers 13-17
Logical EOF pointer 7-40
Loop counter 2-22, 2-23, 2-24
Loop iterations, conditional 2-25
Loop iterations, fixed 2-25
Loop iterations formula 2-22
LOOP statement 2-12, 2-29
LOOP...END LOOP structure 2-27, 2-28, 2-29
Loops 13-11
LWC$ string function 5-19, 5-36, 5-46, 5-48, 5-50, 5-52, 5-54

m

Machine Limits 3-7
Magnitude 3-25
Manual Organization 1-1
Mass Memory Devices, Data Stroage on 13-3
Mass Memory Performance 13-5
Mass storage files 7-1
MASS STORAGE IS statement 17-25
Mass Storage Volume Specification, SRM 17-25
MASS STROAGE IS statement 7-12
MAT binary 4-1
MAT REORDER statement 4-32, 4-36, 4-37
MAT SEARCH, Numeric Comparisons in 4-42
MAT SEARCH statement 4-38, 4-40, 4-42, 4-45, 5-32, 5-33, 5-36
MAT SORT statement 4-34, 4-35, 4-37, 4-43, 5-23, 5-36
MAT statement 4-8, 4-12, 4-26, 5-21
MAT Statement, Using the 4-26
Math Hierarchy 3-10
Mathematical Operations 13-9
Matrices and Vectors 4-47
Matrices, Ill-Conditioned 4-60
Matrices, Singular 4-57
Matrix Functions, Miscellaneous 4-63
Matrix, Identity 4-52
Matrix, Inverse 4-53
Matrix Multiplication 4-47

Matrix, Summing Rows and Columns of a 4-65
Matrix, The Determinant of a 4-59
MAX condition field 4-39, 4-42
MAX function 3-6, 3-19, 5-33
Maximum Values, Minimum and 13-9, 13-13
MAXREAL function 3-16
MC68020 Internal Cache Memory 13-23
MC68881 Floating-Point Math Co-Processor 13-21
Memory, Saving 13-24
MEM_UTILS utility 6-22
Menu, An Expanded Softkey 10-45
MERGE ALPHA statement 10-6
Merging Subprograms 6-29
Methods of Porting 15-2
MIN condition field 4-40, 4-42
MIN function 3-6, 3-19, 5-33
Minimum and Maximum Values 13-9, 13-13
MINREAL function 3-16
Miscellaneous Matrix Functions 4-63
Mnemomic Function Values 8-6
Mode, Compatibility 15-20
Mode, Disabling Display Functions 10-4
Mode, Disabling Printall 10-4
Mode Entries 5-60
Mode Index 5-61
Mode Type 5-61
Modes, Turning Off Unwanted 10-3
Modifying the Source Program(Porting to 4.0) 15-37
Monadic operator 3-12
MOVELINES statement 6-29, 6-30
Moving a Pointer 10-52
Moving EOF Pointers 7-36
Moving the Data Pointer 7-6
MS BIN file 14-3
Multi-character Identifiers, Comments and 13-4
Multiple-Field Numeric Image Specifiers 8-13
Multiple-Line Conditional Segments 2-14
Multiplication, Matrix 4-47
Multiplication With Vectors 4-48
Multiply vs. Add 13-18
Music Editor, A Simple 10-18

n

Nested constructs 2-15
NEXT statement 2-12
NO HEADER statement 7-58
Node 6-35
Non-volatile clock 9-2
Normal program flow 2-30
Null string 5-1
NUM function 5-14
Number builder routine 7-18
Number-Base Conversion 5-34
Numbers, Sequence 5-60
Numeric Arrays 4-1
Numeric Arrays, Searching 4-38
Numeric Comparisons in MAT SEARCH 4-42
Numeric Computation 3-1
Numeric data items 7-18
Numeric Data, Textual 17-6
Numeric Data Types 3-1
Numeric Expressions, Strings in 3-13
Numeric Formats, Internal 3-8
Numeric Image Specifiers 8-12
Numeric Image Specifiers, Examples of 8-13
Numeric Image Specifiers, Multiple-Field 8-13
Numeric-to-String Conversion 5-15
Numerical Functions 3-15

o

OFF CDIAL statement 10-34
OFF CYCLE statement 9-14
OFF DELAY statement 9-14
OFF END statement 7-50
OFF ERROR statement 10-40
OFF KEY statement 2-36, 2-37
OFF KNOB statement 2-36
OFF TIME statement 9-14
OFF-event 2-36
ON CDIAL statement 2-32, 10-33
ON CYCLE statement 2-32, 9-14, 9-15
ON DELAY statement 2-32, 9-14, 9-15
ON END statement 2-32, 7-44, 7-50

ON EOR statement 2-32
ON EOT statement 2-32
ON ERROR branching 11-6
ON ERROR CALL, A Closer Look At 11-11
ON ERROR CALL, Cannot Pass Parameters Using 11-12
ON ERROR Execution at Run-Time 11-6
ON ERROR GOSUB, A Closer Look at 11-8
ON ERROR GOTO, A Closer Look At 11-9
ON ERROR Priority 11-6
ON ERROR RECOVER, A Closer Look At 11-13
ON ERROR statement 2-32, 10-40
ON HIL EXT statement 2-32
ON INTR statement 2-32
ON KBD statement 2-32
ON KEY statement 2-32, 2-33, 6-20, 10-51
ON KNOB "interval" parameter 15-41
ON KNOB statement 2-32, 2-35, 10-32
ON SIGNAL statement 2-32
ON statement 2-20, 2-21
ON TIME statement 2-32, 9-14
ON TIMEOUT statement 2-32, 8-16
ON-event 2-36
ON-event statement 2-30
ON...CALL statement 6-28
One-dimensional array 4-1
One-Dimensional COMPLEX Array in Common 4-6
ON...event statement 2-31
ON...RECOVER statement 6-28
Operand array 4-12
Operations, Mathematical 13-9
Operator, dyadic 3-12
Operator Errors, Anticipating 11-2
Operator Input 10-24
Operator, monadic 3-12
Operator, relational 3-12
Operators 3-12
Operators, Arrays and Arithmetic 4-26
OPTION BASE 6-20
OPTION BASE statement 2-12, 3-3, 4-2, 4-3, 4-32, 5-21
OPTIONAL parameter 6-14
OPTIONAL Parameters 6-13

OPTIONAL/NPAR combination 6-13
Output, Appearance of 8-19
OUTPUT, Random 7-42
OUTPUT, Serial 7-37
OUTPUT statement 2-11, 7-19, 7-23, 7-37
Overhead in ASCII data files, reducing the 7-21
Overview of Human I/O Mechanisms 10-2

p

Page formatter program, writing a 2-8
PaintJet(TM) (HP 3630A) 18-1
Parameter Lists, Formal 6-9
Parameters, Expressions as Pass 3-13
Parameters Lists 6-8
Parameters, OPTIONAL 6-13
Parameters passed by reference 3-5
Parameters passed by value 3-5
Pass parameter list 6-12
Pass Parameter Lists 6-10
Pass Parameters, COM vs. 6-16
Pass Parameters, Expressions as 3-13
Pass Parameters Using ON ERROR CALL, Cannot 11-12
Passing by Value vs. Passing By Reference 6-10
Passing Entire Arrays 4-16
PAUSE statement 2-5
Pausing and Continuing a Program 12-5
PDEV binary 6-29, 6-30
Phasor 3-27
PHYREC CSUB 14-7
PHYREC routine 14-7
PI function 3-17
PIVOT Statement, The 14-19
Planes of a Three-Dimensional REAL Array 4-3
Plotter, Default 14-14
PLOTTER IS Changes 15-39
Pointer, Moving a 10-52
Pointer, Moving the Data 7-6
Pointers 6-35
Pointers, EOF 7-35
Polar Coordinates, Converting from Rectangular to 3-25
Polynomial Evaluations 13-14

Polynomial Order, Time Savings as a Function of 13-16
Porting and Sharing Files 17-1
Porting Considerations, Additional 15-38
Porting LIF Files to SRM 17-24
Porting, Methods of 15-2
Porting to 3.0 14-1
Porting to 5.0 16-1
Porting to Series 300 and 4.0 15-1
Porting Topics Covered 14-2
POS function 2-20, 2-28, 5-12, 5-15
Precision Functions and String Functions, REAL 6-6
Prerun On LOADSUB 14-26
Primary Addresses 8-4
PRINT TAB statement 8-10
PRINT TABXY statement 8-10
PRINTALL IS statement 8-17, 12-14
Printall Mode, Disabling 10-4
Printer Control Characters 8-7
Printer, HP 2225 ThinkJet(TM) 8-1
Printer, HP 2563 Dot-Matrix Impact 8-1
Printer, HP 2565 Dot-Matrix Impact 8-1
Printer, HP 2566 Dot-Matrix Impact 8-1
Printer, HP 2601 Daisy-Wheel Impact 8-1
Printer, HP 2602 Daisy-Wheel Impact 8-1
Printer, HP 2671 Thermal 8-1
Printer, HP 2673 Thermal 8-1
Printer, HP 2686 LaserJet(TM) 8-1
Printer, HP 2932 Dot-Matrix Impact 8-1
Printer, HP 2934 Dot-Matrix Impact 8-1
Printer, HP 82906 Dot-Matrix Impact 8-1
Printer, HP 9876 Thermal 8-1
Printer, Installing, Configuring, and Verifying Your 8-2
PRINTER IS device 4-14, 4-30
PRINTER IS statement 8-2
Printer, The System 8-2
Printer, Using a 8-1
Printer, Using the External 8-7
Printers Supported 8-1
Printing an Entire Array 4-14
Printing Arrays 4-14
Printing Blank Lines 10-6

Priorities, assignable 2-34
Priority, Accent 5-67
Priority, ON ERROR 11-6
Priority Restrictions 9-17
Processor Boards 15-6
PROG Files, Cataloging Individual 7-53
Program Counter 2-7
Program counter 2-2, 2-6
Program Design 6-21
Program flow 2-2
Program, Single-Stepping a 12-10
Program structures, keywords that define 2-12
Program Variables, Using 12-2
Program-interrupt keys 10-24
Program-to-Program Communication 2-41
Programming Additions (BASIC 5.0), General 16-7
Programs, chaining 2-39
Programs, Debugging 12-1
Program/Subprogram Communication 6-8
Prohibited Statements 2-12
Prompting, Displaying and 10-3
PROTECT statement 17-27
PROUND function 3-20
PRT function 3-31

q

Questions, Yes and No 10-43

r

RAD statement 3-17, 3-24, 6-20
Radians 3-17
Random access 7-19, 7-26
Random ENTER 7-48
Random Number Function 3-22
Random OUTPUT 7-42
Random vs. Serial Access 7-26
RANDOMIZE statement 3-22
RANK function 3-16, 4-9
Rate degradation 7-19
Re-Defining an Entire Font 10-15
READ statement 4-12, 7-1, 7-3, 7-4, 7-5

READ Statement to Fill an Entire Array, Using the 4-11
Reading Data From BDAT and HP-UX Files 7-46
Reading String Data From a File 7-46
Reading the Clock 9-3
Read/Write Memory, Data Storage in 13-1
REAL 3-1
REAL and COMPLEX Numbers and Comparisons 11-3
REAL Data Type 3-1
REAL function 3-24, 4-67
REAL Numbers 13-13
REAL Precision Functions and String Functions 6-6
REAL statement 2-12, 4-2, 4-8
REAL value 4-1
Real-Time Clock, Battery-Backed 15-6
Real-time clock, internal 9-14
Real-Time Clock, The 9-1
Real-time programming 2-34
Record Length (BDAT Files Only), Choosing A 7-32
Record Size (BDAT Files Only), Specifying 7-32
Records, Defined 7-31
RECOVER statement 6-19, 6-20, 6-28
RECOVER Statement, Subprograms and the 6-21
Recovery, Scope of Error Trapping and 11-6
Rectangular to Polar Coordinates, Converting from 3-25
Recursion 6-31
REDIM statement 4-7, 4-8, 4-24, 4-25
Redimensioning Arrays 4-24
Redimensioning, Automatic 4-13
Reducing the overhead in ASCII data files 7-21
Reference, Pass by 6-10
References, Cross 12-6
Relational Operations 5-5
Relational operator 3-12
Relay and BNC Video Connectors, The 15-32
REM statement 2-12
REN statement 6-29
Reordering an Array 5-28
Reordering Arrays 4-32
REPEAT statement 2-12, 2-26
Repeat, String 5-18
REPEAT...UNTIL structure 2-22, 2-25, 2-26, 2-27, 2-28, 2-29

RES function 3-31
RESTORE statement 7-6
Restoring the Default Soft Font 10-13
RETURN stack 6-19
RETURN statement 2-7, 11-8
REV$ string function 5-17
Reverse, String 5-17
RND function 3-22
Root 6-35, 6-36
ROTATE function 3-19
Rounding Errors Resulting from Comparisons 3-20
Rounding Functions 3-19
Rounding problem 3-6
Row vector 4-49
Rows and Columns of a Matrix, Summing 4-65
RPLOT with PIVOT 14-19
RPT$ string function 5-18
Rules for Copying Subarrays 4-23
RUN command 6-1
RUN key 2-4
Run-Time, ON ERROR Execution at 11-6
Running Programs, Just Loading and 15-11

S

SAVE statement 6-27, 7-2, 8-18, 13-5
Saving Memory 13-24
Saving Time 13-18
SC function 3-31
Scalar Expressions, Evaluating 3-9
Scope of Error Trapping and Recovery 11-6
Screen, Clearing the 10-5
Screen, Displaying Characters on the 10-8
Screen Width and Height, Determining 10-7
Searching a Three-Dimensional Array 4-43
Searching a Vector 4-38
Searching an Array by Descending Subscripts 4-40
Searching and Sorting 5-22
Searching for Multiple Occurrences 4-45
Searching for Strings 5-30
Searching Numeric Arrays 4-38
Searching String Arrays 5-32

Secondary word DES 4-35
SELECT constructs 2-17
SELECT statement 2-12, 2-18, 2-19, 14-3
Selection 2-10
SEPARATE ALPHA statement 10-6
Sequence Numbers 5-60
Serial access 7-26
Serial ENTER 7-47
Serial Interface 15-7
Serial Interface Configuration, HP 98644 15-12
Serial OUTPUT 7-37
Serial storage of data 7-38
Series 300 Display, Switching Back to the 15-34
Series 300 Hardware, Description of 15-3
Service Routines, Setting Up Error 11-5
SET KEY statement 10-25, 10-27
SET TIME statement 3-29, 9-4
SET TIMEDATE statement 3-29, 9-4, 9-6
Setting Only the Date 9-8
Setting Only the Time 9-6
Setting the Clock 9-4
Setting the clock 3-29
Setting Up Error Service Routines 11-5
SGN function 3-16
Shared Printer, Sending Program Output to a 8-18
Sharing Files, Porting and 17-1
Sharing HFS Discs and Data Files 17-2
SHIFT function 3-19
Shift-in control character 2-27
Shift-out control character 2-27
Simple Branching 2-6
Simulating an Error, Example of 11-15
Simulating Errors (CAUSE ERROR) 11-14
Simultaneous Equations, Solving 4-55
SIN function 3-17
Single Tones, Example of 10-17
Single-Byte Access 7-49
Single-Stepping a Program 12-10
Single-Subscript Substrings 5-6
Singular Matrices 4-57
SINH function 3-18

SIZE function 3-16, 4-9
Soft Font, Restoring the Default 10-13
Soft Font Usage 10-12
Softkey Definitions, Listing Current Typing-Aid 10-25
Softkey Inputs 10-24
Softkey Labels 10-28
Softkey Labels, HP 98203 15-16
Softkey Menu, An Expanded 10-45
Softkeys Programmatically, Defining Typing-Aid 10-25
Softkeys, Subprograms and 6-20
Solving Simultaneous Equations 4-55
Song, Example 10-23
Sorted List, Adding Items to a 5-25
Sorting Arrays 4-34
Sorting by Multiple Keys 5-26
Sorting by Substrings 5-24
Sorting, Searching and 5-22
Sorting to a Vector 4-36, 5-27
Sorting with Automatic REORDER 4-34
Sound, Generating 10-16
Sound Instructions, Arrays of 10-20
SOUND Instructions, Executing Example 10-21
SOUND statement 10-16, 10-20
Source Program(Porting to 4.0), Modifying the 15-37
SPANISH Lexical Order 5-52
Specifier, key 4-44
Specifier, Subarray 4-17
Specifiers, Additional Image 8-15
Specifiers, Numeric Image 8-12
Specifying Record Size (BDAT Files Only) 7-32
Speed Considerations 6-22
Spooler Directories, Writing Files to the 8-18
Spooler directory 8-17
Spooler, Using a 8-17
Spooler, Using SRM Printers through the 8-17
Spooling 8-17
Spooling Using PRINTER IS 8-17
SQR function 3-16
SQRT function 3-16, 3-23
SRM binary 9-1
SRM Directory Paths, Allowing for 17-26

SRM file ... 7-12
SRM File Names, Composition of ... 17-24
SRM File Specifiers ... 17-24
SRM Mass Storage Volume Specification ... 17-25
SRM Passwords vs. LIF Protect Codes ... 17-27
SRM, Porting LIF Files to ... 17-24
SRM Printers through the Spooler, Using ... 8-17
SRM Workstations, Accessing Files Created on Non-Series-200/300 ... 17-29
Starting subscript ... 4-46
Statement Changes ... 14-6
STATUS statement ... 5-43
STATUS/CONTROL Registers (BASIC 5.0), New ... 16-8
Step Functions ... 3-13
STEP key ... 12-10
STOP statement ... 2-4
Storage and Retrieval of Arrays ... 7-5
Storage Format for INTEGER Variables ... 3-8
Storage Format for REAL Variables ... 3-8
Storage in Memory, Character Font ... 10-11
Storage-space efficiency ... 7-26
STORE KEY statement ... 10-26
STORE statement ... 6-27, 7-2, 13-5
Storing and Loading Typing-Aids from Files ... 10-25
Storing Data in Programs ... 7-2
Storing Data in Variables ... 7-2
String ... 5-1
String Array, Cataloging to a ... 7-55
String Arrays ... 5-3
String Arrays and Subarrays, Copying ... 5-21
String Arrays, Searching ... 5-32
String Concatenation ... 5-4
String Data From a File, Reading ... 7-46
String, default dimensioned length of a ... 5-1
String Function, CHR$ ... 5-15, 5-44
String Function, DATE$ ... 9-3
String Function, DVAL$ ... 5-34
String Function, ERRM$ ... 11-8, 11-14
String Function, IVAL$ ... 5-34
String Function, LWC$ ... 5-19, 5-36, 5-46, 5-48, 5-50, 5-52, 5-54
String Function, REV$ ... 5-17
String Function, RPT$ ... 5-18

String Function, SYSTEM$ 5-41, 10-7
String Function, TIME$ 9-3, 9-6
String Function, TRIM$ 5-19
String Function, UPC$ 5-19, 5-36, 5-46, 5-48, 5-50, 5-52, 5-54
String Function, VAL$ 5-15
String Functions 5-17
String Functions, REAL Precision Functions and 6-6
String Image Specifiers 8-14
String Length, Current 5-10
String Length, Maximum 5-10
String Manipulation 5-1
String Repeat 5-18
String Reverse 5-17
String Storage 5-2
String, Trimming a 5-19
String variable 5-1
String variable's length header 7-20
String-Related Functions 5-10
String-to-Numeric Conversion 5-13
Strings, Evaluating Expressions Containing 5-4
Strings in Numeric Expressions 3-13
Strings, Searching for 5-30
Strings, Textual 17-9
Stroage, Data 13-1
SUB statement 2-12, 6-4, 6-8, 6-28, 6-29, 6-30, 6-31
Subarray, Copying a Subarray into another 4-21
Subarray Specifier 4-17
Subarray specifier examples 4-18
Subarrays, Copying 4-16
Subarrays, Copying String Arrays and 5-21
Subarrays, Rules for Copying 4-23
SUBEND statement 2-12, 6-27, 6-30
SUBEXIT statement 6-30
Subprogram 2-7
Subprogram and User-Defined Function Names 6-5
Subprogram, Calling and Executing a 6-4
Subprogram, Difference Between a User-Defined Function and a 6-6
Subprogram Entry Execution Speed 6-23
Subprogram Libraries, Using 6-24
Subprogram Location 6-5
Subprograms, A Closer Look at 6-4

Subprograms and Softkeys 6-20
Subprograms and Subroutines, Differences Between 6-5
Subprograms and the RECOVER Statement 6-21
Subprograms and User-Defined Functions 6-1
Subprograms at Once, Loading Several 6-25
Subprograms, Benefits of 6-3
Subprograms, Calling 12-4
Subprograms, Deleting 6-27, 6-29
Subprograms from the Keyboard, Calling 6-22
Subprograms in a PROG File, Listing the 6-24
Subprograms, Inserting 6-28
Subprograms, Loading 6-24
Subprograms, Merging 6-29
Subprograms One at a Time, Loading 6-25
Subprograms Prior to Execution, Loading 6-26
Subroutine 2-7
Subscript expression 4-17
Subscript range 4-17
Subscript, starting 4-46
Substring Position 5-11
Substrings 5-6
Substrings, Double-Subscript 5-7
Substrings, Single-Subscript 5-6
Substrings, Sorting by 5-24
Subtree 6-36
SUM function 3-16
Summing Rows and Columns of a Matrix 4-65
Summing the Elements in an Array 4-29
Suppressing the Catalog Header 7-58
SWEDISH Lexical Order 5-54
Switching Back to the Series 300 Display 15-34
Symbol table 13-1
symbol table 13-1
System key sequences 10-31
System Printer, The 8-2
System Sector, BDAT File 7-31
SYSTEM$ string function 5-41, 10-7
SYSTEM$("SYSTEM ID") Values, New 15-38

t

TAB function 10-6
Tablet DIGITIZE, Graphics 14-15
TAN function 3-17
TANH function 3-18
Textual Numeric Data 17-6
Textual Strings 17-9
Three-Dimensional Array, Searching a 4-43
Three-Dimensional INTEGER Array 4-6
Time and Date Functions 3-29
Time Format, Clock 9-4
TIME function 3-29, 9-6
Time of Day 9-16
Time, Saving 13-18
Time Savings as a Function of Polynomial Order 13-16
Time, Setting Only the 9-6
TIME$ string function 9-3, 9-6
TIMEDATE function 3-29, 9-3, 9-6
Timing, Interval 9-13
Token table 13-1
token table 13-1
Tones, Example of Single 10-17
Top-Down Design 6-33, 6-36
TRACE ALL statement 12-12, 12-13
TRACE OFF statement 12-16
TRACE PAUSE statement 12-15
Tracing 12-12
Transpose Function 4-63
Trapping and Recovery, Scope of Error 11-6
Trapping EOF and EOR Conditions 7-50
Trapping Errors with BASIC Programs 11-5
Trapping (OFF ERROR), Disabling Error 11-6
Tree structure 6-41
Trigonometric Functions 3-17
Trigonometric Mode, COMPLEX Arguments and the 3-24
TRIM$ string function 5-19
Trimming a String 5-19
TRN function 4-63
Turning Off Unwanted Modes 10-3
Two-dimensional 4-2
Two-Dimensional COMPLEX Array 4-5

Two-Dimensional REAL Array 4-4
Type Conversions 13-13
Type Conversions, Implicit 3-4
Typing-aid keys 10-24
Typing-Aid Softkey Definitions, Listing Current 10-25
Typing-Aid Softkeys Programmatically, Defining 10-25
Typing-Aids from Files, Storing and Loading 10-25
Typing-aids keys 10-24

U

Unreferenced entries 12-8
UNTIL statement 2-12
Unused Entries 12-8
UPC$ string function 5-19, 5-36, 5-46, 5-48, 5-50, 5-52, 5-54
User-defined formats 7-27
User-defined LEXICAL ORDER 5-56
Utilities (BASIC 5.0), New 16-3

V

VAL function 5-13
VAL$ function 7-23
VAL$ string function 5-15
Value, Pass by 6-10
Variable and Array Initialization 13-4
Variable Initialization 6-20
Variables, Assigning 3-4
Variables, Declaring 3-3
Variables, keywords used in the declaration of 2-12
Variables, Using Program 12-2
Vector, Searching a 4-38
Vector, Sorting to a 4-36, 5-27
Vectors, Matrices and 4-47
Vectors, Multiplication With 4-48
Verifying Your Printer, Installing, Configuring, and 8-2
VIEWPORT Statement, The 14-15
VME Interface (HP 98646A) 18-1
Volatile real-time clocks 9-2
Volume Specification, SRM Mass Storage 17-25

w

WAIT statement 2-5
Week, Day of the 9-13
WHILE statement 2-12, 2-26
WHILE...END structure 2-22
WHILE...END WHILE structure 2-27, 2-29
Width and Height, Determining Screen 10-7
Writing a page formatter program 2-8
Writing Data 7-37
Writing Files to the Spooler Directories 8-18

x

XREF statement 12-6

y

Yes and No Questions 10-43

MANUAL COMMENT CARD

BASIC 5.0/5.1
Programming Techniques

HP Part Number 98613-90012 11/87

Please help us improve this manual. Circle the numbers in the following statement that best indicate how useful you found this manual. Then add any further comments in the spaces below. **In appreciation of your time, we will enter your name in a quarterly drawing for an HP calculator.** Thank you.

The information in this manual:

Is poorly organized	1	2	3	4	5	Is well organized
Is hard to find	1	2	3	4	5	Is easy to find
Doesn't cover enough	1	2	3	4	5	Covers everything
Has too many errors	1	2	3	4	5	Is very accurate

fold ——

Particular pages with errors? ______________________________

____

Comments: ______________________________

____

____

____

____

____

____

Name: ______________________________

Job Title: ______________________________

Company: ______________________________

Address: ______________________________

____

☐ Check here if you wish a reply.

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 37 LOVELAND, COLORADO

POSTAGE WILL BE PAID BY ADDRESSEE

Hewlett-Packard Company
Attn: Customer Documentation
3404 East Harmony Road
Fort Collins, Colorado 80525

HEWLETT PACKARD

HP Part Number
98613-90012

Microfiche No. 98613-99012
Printed in U.S.A. 11/87

98613-90665
For Internal Use Only