ManualsPlus.com

HP Instrument BASIC User's Handbook

HP 8711A RF Network Analyzer

HP Part No. 08711-90112
Printed in USA July 1993

ManualsPlus.com

Notice.

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

Hewlett-Packard makes no warranty of any kind with regard to this material, including but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

© Copyright Hewlett-Packard Company 1992, 1993
All Rights Reserved. Reproduction, adaptation, or translation without prior written permission is prohibited, except as allowed under the copyright laws.
1400 Fountaingrove Parkway, Santa Rosa CA, 95403-1799, USA

ManualsPlus.com

Contents

ManualsPlus.com

ManualsPlus.com

ManualsPlus.com

ManualsPlus.com

1

Manual Overview

Introduction

This manual presents the concepts of computer interfacing that are relevant to programming in HP Instrument BASIC. Note that not all features described in this manual may be implemented on your instrument. Please consult your instrument-specific manual for a description of implemented features. The topics presented herein will increase your understanding of interfacing the host instrument and external devices and computers with HP Instrument BASIC programs.

Manual Organization

This manual is organized by topics and is designed as a learning tool, not a reference. The text is arranged to focus your attention on interfacing concepts rather than to present only a serial list of the HP Instrument BASIC language I/O statements. Once you have read this manual and are familiar with the general and specific concepts involved, you can use either this manual or the *HP Instrument BASIC Language Reference* when searching for a particular detail of how a statement works.

This manual is designed for easy access by both experienced programmers and beginners.

Beginners	may want to begin with Chapter 2, "Interfacing Concepts", before reading about general or interface-specific techniques.
Experienced programmers	may decide to go directly to the chapter in your instrument-specific manual that describes the particular interface to be used. It is also usually helpful to become familiar with display and keyboard I/O operations, since these are helpful in seeing results while testing I/O programs. If you need more background as you read about a particular topic, consult chapters 3 through 6 for a detailed explanation.

The brief descriptions in the next section will help you determine which chapters you will need to read for your particular application.

ManualsPlus.com

Chapter Previews

This manual is intended to provide background and tutorial information for programmers who have not written HP Instrument BASIC I/O programs before. It presents topics and programming techniques applicable to all interfaces.

Chapter 2: Interfacing Concepts

This chapter presents a brief explanation of relevant interfacing concepts and terminology. This discussion is especially useful for beginners as it covers much of the "why" and "how" of interfacing. Experienced programmers may also want to review this material to better understand the terminology used in this manual.

Chapter 3: Directing Data Flow

This chapter describes how to specify which instrument resource is to send data to or receive data. The use of device selectors, string variable names, and "I/O path names" in I/O statements are described.

Chapter 4: Outputting Data

This chapter presents methods of outputting data to devices. All details of this process are discussed, and several examples of free-field output and output using images are given. Since this chapter completely describes outputting data to devices, you may only need to read the sections relevant to your application.

Chapter 5: Entering Data

This chapter presents methods of entering data from devices. All details of this process are discussed, and several examples of free-field enter and enter using images are given. As with Chapter 4, you may only need to read sections of this chapter relevant to your application.

Chapter 6: I/O Path Attributes

This chapter presents several powerful capabilities of the I/O path names provided by the BASIC language system. Interfacing to devices is compared to interfacing to mass storage files, and the benefits of using the same statements to access both types of resources are explained. This chapter is also highly recommended to all readers.

Specific Interfaces

Since each host instrument for HP Instrument BASIC implements the display, keyboard and other interfaces in slightly different manners, this manual does not cover the operation of interfaces. For specific details on the operation of interfaces with HP Instrument BASIC, consult the instrument-specific manual for your host instrument.

ManualsPlus.com

2

Interfacing Concepts

This chapter describes the functions and requirements of interfaces between the host instrument and its resources. Concepts in this chapter are presented in an informal manner. *All* levels of programmers can gain useful background information that will increase their understanding of the *why* and *how* of interfacing.

Terminology

These terms are important to your understanding of the text of this manual. The purpose of this section is to make sure that our terms have the same meanings.

computer — is herein defined to be the processor, its support hardware, and the HP Instrument BASIC-language system of the *host instrument*; together these system elements *manage* all computer resources.

hardware — describes both the electrical connections and electronic devices that make up the circuits within the computer; any piece of hardware is an actual physical device.

software — describes the user-written, BASIC-language programs.

firmware — refers to the preprogrammed, machine-language programs that are invoked by BASIC-language statements and commands. As the term implies, firmware is not usually modified by BASIC users. The machine-language routines of the operating system are firmware programs.

computer resource — is herein used to describe all of the "data-handling" elements of the system. Computer resources include: internal memory, display, keyboard, and disc drive, and any external devices that are under computer control.

I/O — is an acronym that comes from "Input and Output"; it refers to the process of copying data to or from computer memory.

output — involves moving data from computer memory to another resource. During output, the **source** of data is computer memory and the **destination** is any resource, including memory.

input — is moving data from a resource to computer memory; the source is any resource and the destination is a variable in computer memory. *Inputting data is also referred to as "entering data" in this manual* for the sake of avoiding confusion with the INPUT statement.

bus — refers to a common group of hardware lines that are used to transmit information between computer resources. The computer communicates directly with the internal resources through the data and control buses.

ManualsPlus.com

computer backplane is an extension of these internal data and control buses. The computer communicates indirectly with the external devices through interfaces connected to the backplane hardware.

Why Do You Need an Interface?

The primary function of an interface is to provide a communication path for data and commands between the computer and its resources. Interfaces act as intermediaries between resources by handling part of the "bookkeeping" work, ensuring that this communication process flows smoothly. The following paragraphs explain the need for interfaces.

First, even though the computer bus is driven by electronic hardware that generates and receives electrical signals, this hardware was not designed to be connected directly to external devices. The internal hardware has been designed with specific electrical logic levels and drive capability in mind.

Second, you cannot be assured that the connectors of the computer and peripheral are compatible. In fact, there is a good probability that the connectors may not even mate properly, let alone that there is a one-to-one correspondence between each signal wire's function.

Third, assuming that the connectors and signals are compatible, you have no guarantee that the data sent will be interpreted properly by the receiving device. Some peripherals expect single-bit serial data while others expect data to be in 8-bit parallel form.

Fourth, there is no reason to believe that the computer and peripheral will be in agreement as to when the data transfer will occur; and when the transfer does begin, the transfer rates will probably not match.

As you can see, interfaces have a great responsibility to oversee the communication between computer and its resources.

Electrical and Mechanical Compatibility

Electrical compatibility must be ensured before any thought of connecting two devices occurs. Often the two devices have input and output signals that do not match; if so, the interface serves to match the electrical levels of these signals before the physical connections are made.

Mechanical compatibility simply means that the connector plugs must fit together properly. The interfaces connect with the computer buses. The peripheral end of the interfaces have connectors that match those on peripherals.

Data Compatibility

Just as two people must speak a common language, the computer and peripheral must agree upon the form and meaning of data before communicating it. As a programmer, one of the most difficult requirements to fulfill before exchanging data is that the format and meaning of the data being sent is identical to that anticipated by the receiving device. Even though some interfaces format data, most do not; most interfaces merely move data to or from computer memory. The computer must make the necessary changes, if any, so that the receiving device gets meaningful information.

ManualsPlus.com

Timing Compatibility

Since all devices do not have standard data-transfer rates, nor do they always agree as to when the transfer will take place, a consensus between sending and receiving device must be made. If the sender and receiver can agree on both the transfer rate and beginning point (in time), the process can be made readily.

If the data transfer is not begun at an agreed-upon point in time and at a known rate, the transfer must proceed one data item at a time with acknowledgement from the receiving device that it has the data and that the sender can transfer the next data item; this process is known as a "handshake." Both types of transfers are utilized with different interfaces and both will be fully described as necessary.

Additional Interface Functions

Another powerful feature of some interfaces is to relieve the computer of low-level tasks, such as performing data-transfer handshakes. This distribution of tasks eases some of the computer's burden and also decreases the otherwise-stringent response-time requirements of external devices. The actual tasks performed by each type of interface vary widely and are described in the next section of this chapter.

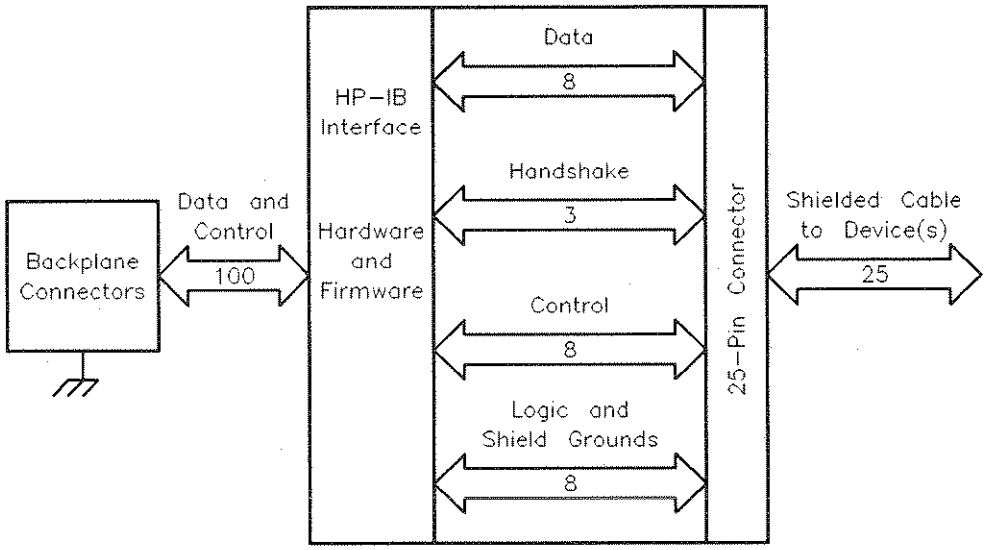
ManualsPlus.com

Interface Overview

Now that you see the need for interfaces, you should see what kinds of interfaces are available for the computer. Each of these interfaces is specifically designed for specific methods of data transfer; each interface's hardware configuration reflects its function.

The HP-IB Interface

This interface is Hewlett-Packard's implementation of the IEEE-488 1978 Standard Digital Interface for Programmable Instrumentation. The acronym "HP-IB" comes from Hewlett-Packard Interface Bus, often called the "bus".



Block Diagram of the HP-IB Interface

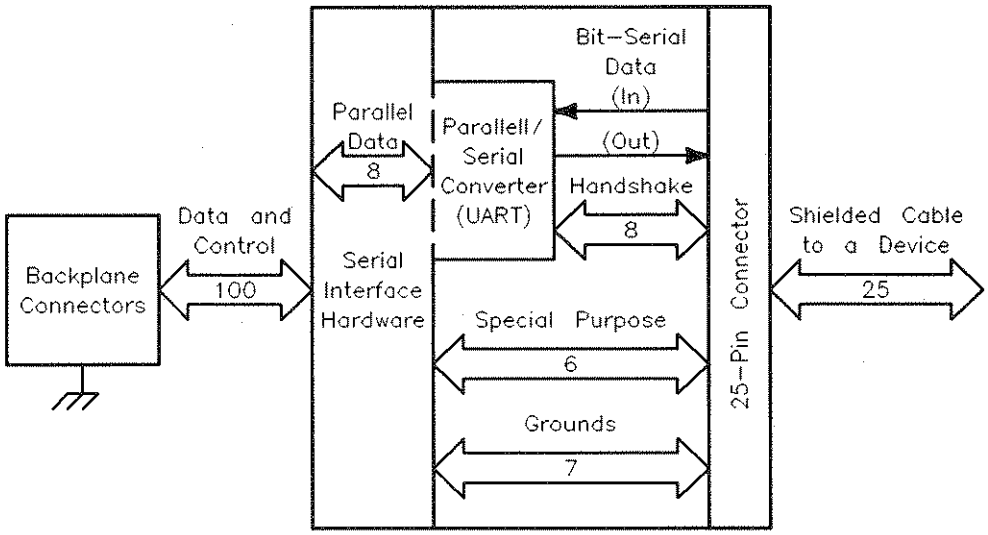
The HP-IB interface fulfills all four compatibility requirements (hardware, electrical, data, and timing) with no additional modification. Just about all you need to do is connect the interface cable to the desired HP-IB device and begin programming. All resources connected to the computer through the HP-IB interface must adhere to this IEEE standard.

The "bus" is somewhat of an independent entity; it is a communication arbitrator that provides an organized protocol for communications between several devices. The bus can be configured in several ways. The devices on the bus can be configured to act as senders or receivers of data and control messages, depending on their capabilities.

ManualsPlus.com

The RS-232C Serial Interface

The serial interface changes 8-bit parallel data into bit-serial information and transmits the data through a two-wire (usually shielded) cable; data is received in this serial format and is converted back to parallel data. This use of two wires makes it more economical to transmit data over long distances than to use 8 individual lines.



Block Diagram of the Serial Interface

Data is transmitted at several programmable rates using either a simple data handshake or no handshake at all. The main use of this interface is in communicating with simple devices.

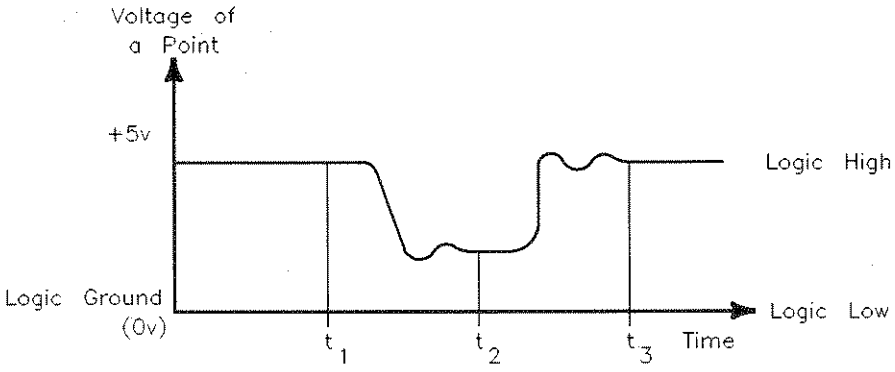
ManualsPlus.com

Data Representations

As long as data is only being used internally, it really makes little difference how it is represented; the computer always understands its own representations. However, when data is to be moved to or from an external resource, the data representation is of paramount importance.

Bits and Bytes

Computer memory is no more than a large collection of individual bits (*bi*nary digi*ts*), each of which can take on one of two logic levels (high or low). Depending on how the computer interprets these bits, they may mean on or not on (off), true or not true (false), one or zero, busy or not busy, or any other bi-state condition. These logic levels are actually voltage levels of hardware locations within the computer. The following diagram shows the voltage of a point versus time and relates the voltage levels to logic levels.



Voltage and Positive-True Logic

In some cases, you want to determine the state of an individual bit (of a variable in computer memory, for instance). The logical binary functions (BIT, BINCMP, BINIOR, BINEOR, BINAND, ROTATE, and SHIFT) provide access to the individual bits of data.

In most cases, these individual bits are not very useful by themselves, so the computer groups them into multiple-bit entities for the purpose of representing more complex data. Thus, all data in computer memory are somehow represented with binary numbers.

The computer's hardware accesses groups of sixteen bits at one time through the internal data bus; this size group is known as a **word**. With this size of bit group, 65 536 ($65\,536=2^{16}$) different bit patterns can be produced. The computer can also use groups of eight bits at a time; this size group is known as a **byte**. With this smaller size of bit group, 256 ($256=2^{8}$) different patterns can be produced. How the computer and its resources interpret these combinations of ones and zeros is very important and gives the computer all of its utility.

ManualsPlus.com

Representing Numbers

The following binary weighting scheme is often used to represent numbers with a single data byte. Only the non-negative integers 0 through 255 can be represented with this particular scheme.

Most-Significant Bit Least-Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	0	0	1	0	1	1	0
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

Notice that the value of a 1 in each bit position is equal to the power of two of that position. For example, a 1 in the 0th bit position has a value of 1 ($1=2^0$), a 1 in the 1st position has a value of 2 ($2=2^1$), and so forth. The number that the byte represents is then the total of all the individual bit's values.

$0 \times 2^0 = 0$
$1 \times 2^1 = 2$
$1 \times 2^2 = 4$ Number represented =
$0 \times 2^3 = 0$
$1 \times 2^4 = 16$ $2 + 4 + 16 + 128 = 150$
$0 \times 2^5 = 0$
$0 \times 2^6 = 0$
$1 \times 2^7 = 128$

The preceding representation is used by the "NUM" function when it interprets a byte of data. The next section explains why the character "A" can be represented by a single byte.

```
100  Number=NUM("A")
110  PRINT " Number = ";Number
120  END
```

prints

```
Number = 65
```

Representing Characters

Data stored for humans is often alphanumeric-type data. Since less than 256 characters are commonly used for general communication, a single data byte can be used to represent a character. The most widely used character set is defined by the ASCII standard. ASCII stands for "American Standard Code for Information Interchange". This standard defines the correspondence between characters and bit patterns of individual bytes. Since this standard only defines 128 patterns (bit 7 = 0), 128 additional characters are defined by the computer (bit 7 = 1). The entire set of the 256 characters on the computer is hereafter called the "extended ASCII" character set.

When the CHR$ function is used to interpret a byte of data, its argument must be specified by its binary-weighted value. The single (extended ASCII) character returned corresponds to the bit pattern of the function's argument.

ManualsPlus.com

```
100   Number=65                ! Bit pattern is "01000001"
110   PRINT " Character is ";
120   PRINT CHR$(Number)
130   END
```

prints

```
Character is A
```

The I/O Process

When using statements that move data between memory and internal computer resources, you do not usually need to be concerned with the details of the operations. However, you may have wondered how the computer moves the data. This section describes I/O operations regarding how the computer outputs and enters data.

I/O Statements and Parameters

The I/O process begins when an I/O statement is encountered in a program. The computer first determines the type of I/O statement to be executed (such as, OUTPUT, ENTER, USING, etc.) Once the type of statement is determined, the computer evaluates the statement's parameters.

Specifying a Resource

Each resource must have a unique specifier that allows it to be accessed to the exclusion of all other resources connected to the computer. The methods of uniquely specifying resources (output destinations and enter sources) are device selectors, string variable names, and I/O path names. These specifiers are further described in the next chapter.

For instance, before executing an OUTPUT statement, the computer first evaluates the parameter that specifies the destination resource. The source parameter of an ENTER statement is evaluated similarly.

```
OUTPUT Dest_parameter;Source_item

ENTER Sourc_parameter;Dest_item
```

Data Handshake

Each byte (or word) of data is transferred with a procedure known as a data-transfer handshake (or simply "handshake"). It is the means of moving one byte of data at a time when the two devices are not in agreement as to the rate of data transfer or as to what point in time the transfer will begin. The steps of the handshake are as follows:

1. The sender signals to get the receiver's attention.
2. The receiver acknowledges that it is ready.
3. A data byte (or word) is placed on the data bus.
4. The receiver acknowledges that it has gotten the data item and is now busy. No further data may be sent until the receiver is ready.
5. Repeat these steps if more data items are to be moved.

ManualsPlus.com

3

Directing Data Flow

Data can be moved between computer memory and several resources. These resources include:

- Computer memory
- Internal and external devices
- Mass storage files

This chapter describes in general terms how devices and string variables are specified in I/O statements. Each of these topics is covered in more detail in subsequent chapters. This chapter also describes the use of I/O pathnames in specifying devices for later use in I/O statements.

Specifying a Resource

Each resource must have a specifier that allows it to be accessed to the exclusion of all other computer resources. String variables are specified by variable name, while devices can be specified by either their device selector or a data type known as an I/O path name. This section describes how to specify these resources in OUTPUT and ENTER statements.

String-Variable Names

Data is moved to and from string variables by specifying the string variable's name in an OUTPUT or ENTER statement. Examples of each are shown below:

```
200  OUTPUT To_string$;Data_out$; ! ";" suppresses CR/LF.
240  ENTER From_string$;To_string$
```

Data is always copied to the destination string (or from the source string) beginning at the first position of the variable; subscripts cannot be used to specify any other beginning position within the variable.

Formatted String I/O

The use of outputting to and entering from string variables is a very powerful method of buffering data to be output to other resources. With OUTPUT and ENTER statements that use images, the data sent to the string variables can be explicitly formatted before being sent to (or while being received from) the variable.

ManualsPlus.com

Device Selectors

Devices include an internal CRT, keyboard, external printers and instruments, and all other physical entities that can be connected to the computer through an interface. Each interface has a unique number by which it is identified, known as its **interface select code.**

In order to send data to or receive data from a device, merely specify the select code of its interface in an OUTPUT or ENTER statement. Examples of using select codes to access devices are shown below.

```
OUTPUT 1;"Data to CRT"
ENTER CRT;Crt_line$

HPib_device=722
OUTPUT 722;"F1R1"
ENTER Hpib_device;Reading
```

The following pages explain select codes and device selectors.

Select Codes of Built-In Interfaces

The internal devices are accessed with the following, permanently-assigned interface select codes.

Note Some host instruments may not contain all of the following interfaces.

Select Codes of Built-In Devices

Built-In Interface/Device	Permanent Select Code
Alpha Display	1
Keyboard	2
Built-in HP-IB interface	7
Built-in serial interface	9

The host instrument may have other built-in interfaces. See your instrument-specific HP Instrument BASIC manual for information regarding these interfaces and their select codes.

HP-IB Device Selectors

Each device on the HP-IB interface has a **primary address** by which it is uniquely identified; each address must be unique so that only one device is accessed when one address is specified. The device selector is then a combination of the interface select code and the device's address. Some examples are shown below.

ManualsPlus.com

HP-IB Device Selector Examples

Device Location	Device Selector	Example I/O Statement
interface select code 7, primary address 22	722	`OUTPUT 722;"Data" ENTER 722;Number`
interface select code 10, primary address 01	1001	`OUTPUT 1001;"Data" ENTER 1001;Number`

I/O Paths

All data entered and output via an interface to files or devices is moved through an "I/O Path." The I/O paths to devices and mass storage files can be assigned special names called **I/O path names**. I/O paths to strings cannot use I/O path names. The next section describes how to use I/O path names along with the benefits of using them.

I/O Path Names

An I/O path name is a data type that describes an I/O resource. With HP Instrument BASIC, you can assign I/O path names to either a device or a data file on a mass storage device. The following examples show how this is done.

Devices `ASSIGN @Device TO 722`

Files `ASSIGN @File TO "MyFile"`

Once assigned, the I/O path names can be used in place of the device selectors to specify the resource with which communication is to take place. For example:

`ASSIGN @Display TO 1`	Assigns the I/O path name @Display to the CRT.
`OUTPUT @Display;"Data"`	Sends characters to the display.
`ASSIGN @Printer TO 701`	Assigns @Printer to HP-IB device 701.
`OUTPUT @Printer;"Data"`	Sends characters to the printer.
`ASSIGN @Gpio TO 12`	Assigns @Gpio to the interface at select code 12.
`ENTER @Gpio;A_number`	Enters one numeric value from the interface.

Note HP Instrument BASIC does not support assigning an I/O path name to multiple devices.

Since an I/O path name is a data type, a fixed amount of memory is allocated for the variable, similar to the manner in which memory is allocated to other program variables (integer, real and string). This I/O path information is only accessible to the context in which it was allocated, unless it is passed as a parameter or appears in the proper COM statements.

ReAssigning I/O Path Names

If an I/O path name already assigned to a resource is to be reassigned to another resource, the preceding form of the ASSIGN statement is also used. The resultant action is that the the

ManualsPlus.com

I/O path name to the device is implicitly closed. A new assignment is then made just as if the first assignment never existed.

```
100   ASSIGN @Printer TO 1     ! Initial assignment.
110   OUTPUT @Printer;"Data1"
120   !
130   ASSIGN @Printer TO 701   ! 2nd ASSIGN closes 1st
140   OUTPUT @Printer;"Data2" ! and makes a new assignment.
150   PAUSE
160   END
```

The result of running the program is that "Data1" is sent to the CRT, and "Data2" is sent to HP-IB device 701.

Closing I/O Path Names

A second use of the ASSIGN statement is to *explicitly close* the name assigned to an I/O path. For example, to close the path name @Printer you would use the following statement:

```
ASSIGN @Printer TO *
```

After executing this statement for a particular I/O path name, the name cannot be used in subsequent I/O statements until it is reassigned.

I/O Path Names in Subprograms

When a subprogram (either a SUB subprogram or a user-defined function) is called, the "context" is changed to that of the called subprogram. The statements in the subprogram only have access to the data of the new context. Thus, in order to use an I/O path name in any statement within a subprogram, one of the following conditions must be true:

- The I/O path name must already be assigned within the context (i.e., the same instance of the subprogram)
- The I/O path name must be assigned in another context and passed to this context by reference (i.e., specified in both the formal-parameter and pass-parameter lists)
- The I/O path name must be declared in a variable common (with COM statements) and already be assigned within a context that has access to that common block

The following paragraphs and examples further describe using I/O path names in subprograms.

Assigning I/O Path Names Locally Within Subprograms

Any I/O path name can be used in a subprogram if it has first been assigned to an I/O path within the subprogram. A typical example is shown below.

```
10   CALL Subprogram_x
20   END
30   !
40   SUB Subprogram_x
50   ASSIGN @Log_device TO 1 ! CRT.
60   OUTPUT @Log_device;"Subprogram"
70   SUBEND
```

ManualsPlus.com

When the subprogram is exited, all I/O path names assigned locally within the subprogram are automatically closed. If the program (or subprogram) that called the exited subprogram attempts to use the I/O path name, an error results. An example of this closing local I/O path names upon return from a subprogram is shown below.

```
10   CALL Subprogram_x
11   OUTPUT @Log_device;"Main" ! inserted line
20   END
30   !
40   SUB Subprogram_x
50   ASSIGN @Log_device TO 1 ! CRT.
60   OUTPUT @Log_device;"Subprogram"
70   SUBEND
```

When the above program is run, error 177, *Undefined I/O path name*, occurs in line 11.

Each context has its own set of local variables. These variables are not automatically accessible to any other context. Consequently, if the same I/O path name is assigned to I/O paths in separate contexts, the assignment local to the context is used while in that context. Upon return to the calling context, any I/O path names accessible to this context remain assigned as before the context was changed.

```
1    ASSIGN @Log_device to 701         ! Inserted line
2    OUTPUT @Log_device;"First Main"   ! Inserted line
10   CALL Subprogram_x
11   OUTPUT @Log_device;"Second Main" ! Changed line
20   END
30   !
40   SUB Subprogram_x
50   ASSIGN @Log_device TO 1 ! CRT.
60   OUTPUT @Log_device;"Subprogram"
70   SUBEND
```

The results of the above program are that the outputs “First Main” and “Second Main” are directed to device 701, while the output “Subprogram” is directed to the CRT. Notice that the original assignment of @Log_device made to interface select code 1 was local to the subprogram.

Passing I/O Names as Parameters

I/O path names can be used in subprograms if they are assigned and have been passed to the called subprogram by reference; they cannot be passed by value. The I/O path names(s) to be used must appear in both the pass-parameter and formal-parameter lists.

```
1    ASSIGN @Log_device to 701
2    OUTPUT @Log_device;"First Main"
10   CALL Subprogram_x(@Log_device)    ! Add pass parameter
11   OUTPUT @Log_device;"Second Main"
20   END
30   !
40   SUB Subprogram_x(@Log)            ! Add formal parameter
50   ASSIGN @Log TO 1 ! CRT.
60   OUTPUT @Log;"Subprogram"
70   SUBEND
```

Upon returning to the calling routine, any changes made to the assignment of the I/O path name passed by reference are maintained; the assignment local to the calling context is not restored as in the preceding example, since the I/O path name is accessible to both contexts.

ManualsPlus.com

In this example, @Log_device remains assigned to interface select code 1; thus, "Subprogram" and "Second Main" are both directed to the CRT.

Declaring I/O Path Names in Common

An I/O path name can also be accessed by a subprogram if it has been declared in a COM statement (labeled or unlabeled) common to calling and called contexts, as shown in the following example.

```
1    COM @Log_device                  ! Insert COM statement
3    ASSIGN @Log_device to 701
4    OUTPUT @Log_device;"First Main"
10   CALL Subprogram_x                ! Parameters not necessary
11   OUTPUT @Log_device;"Second Main"
20   END
30   !
40   SUB Subprogram_x                 ! Parameters not necessary
41   COM @ Log_device                 ! Insert COM statement
50   ASSIGN @Log_device TO 1 ! CRT.
60   OUTPUT @Log_device;"Subprogram"
70   SUBEND
```

If an I/O path name is common is modified in any way, the assignment is changed for all subsequent contexts; the original assignment is not "restored" upon exiting the subprogram. In this example, "First Main" is sent to the HP-IB device 701, but "Subprogram" and "Second Main" are both directed to the CRT. This is identical to the preceding action when the I/O path name was passed by reference.

Benefits of Using I/O Path Names

Assigning names to I/O paths provide improvements in performance and additional capabilities over using device selectors. These advantages fall in the following areas:

- execution speed
- redirecting data to or from other destinations
- access to mass storage files
- attribute control

Execution Speed

When a device selector is used in an I/O statement to specify the I/O path to a device, first the numeric expression must be evaluated, then the corresponding attributes of the I/O path must be determined before the I/O path can be used. If an I/O path name is specified in an OUTPUT or ENTER statement, all of this information has already been determined at the time the I/O path name was assigned. Thus, an I/O statement containing an I/O path name executes slightly faster than using the corresponding I/O statement containing a device selector (for the same set of source-list expressions).

ManualsPlus.com

Redirecting Data

Using numeric-variable device selectors, as with I/O path names, allows a single statement to be used to move data between the computer and several devices. Simple examples of redirecting data in this manner are shown in the following programs.

Example of Re-Directing with Device Selectors

```
100   Device=1
110   GOSUB Data_out
       .
       .
200   Device=701
210   GOSUB Data_out
       .
       .
410   Data_out: OUTPUT Device;Data$
420             RETURN
```

Example of Re-Directing with I/O Path Names

```
100   ASSIGN @Device TO 1
110   GOSUB Data_out
       .
       .
200   ASSIGN @Device TO 9
210   GOSUB Data_out
       .
       .
410   Data_out: OUTPUT @Device;Data$
420             RETURN
```

The preceding two methods of redirecting data execute in approximately the same amount of time.

Access to Mass Storage Files

The third advantage of using I/O path names is that device selectors cannot be used to direct data to or from mass storage files. Therefore, I/O path names are the only access to files. If the data is ever to be directed to a file, you must use I/O path names.

Attribute Control

I/O paths have certain "attributes" that control how the system handles data sent through the I/O path. For example, the FORMAT attribute possessed by an I/O path determines which data representation will be used by the path during communications. If the path possesses the attribute of FORMAT ON, the ASCII data representation will be used. This is the default attribute automatically assigned by the computer when I/O path names are assigned to device selectors. If the I/O path possesses the attribute of FORMAT OFF, the internal data representation is used; this is the default format for BDAT files. Further details of these and additional attributes are discussed in the "I/O Path Attributes" chapter.

The final factor that favors using I/O path names is that you can control which attribute(s) are to be assigned to the I/O path. Attributes can be attached to an I/O path name when it is assigned to a device (via the ASSIGN statement) and can specify data representation (ASCII or internal) as well as the end-of-line sequence for all data using the path. Details of these attributes are discussed in the "I/O Path Attributes" chapter.

ManualsPlus.com

ManualsPlus.com

4

Outputting Data

Introduction

This chapter describes the topic of outputting data to devices; outputting data to string variables, and mass storage files is described in the "I/O Path Attributes" chapter of this manual, in "Data Storage and Retrieval", chapter 7 of *HP Instrument BASIC Programming Techniques*.

There are two general types of output operations. The first type, known as "free-field outputs", use the HP Instrument BASIC's default data representations. The second type provides precise control over each character sent to a device by allowing you to specify the exact "image" of the ASCII data to be output.

Free-Field Outputs

Free-field outputs are invoked when the following types of OUTPUT statements are executed.

Examples

```
OUTPUT @Device;3.14*Radius^2

OUTPUT Printer;"String data";Num_1

OUTPUT 9;Test,Score,Student$

OUTPUT Escape_code$;CHR$(27)&"&A1S";
```

The Free-Field Convention

The term "free-field" refers to the number of characters used to represent a data item. During free-field outputs, HP Instrument BASIC does *not* send a *constant* number of ASCII characters for each type of data item, as is done during "fixed-field outputs" which use images (described later). Instead, a special set of rules is used that govern the number and type of characters sent for each source item. The rules used for determining the characters output for numeric and string data are described in the following paragraphs.

Standard Numeric Format

The default data representation for devices is to use ASCII characters to represent numbers. The ASCII representation of each expression in the source list is generated during free-field output operations. Even though all REAL numbers have 15 (and INTEGERs can have up to 5) significant decimal digits of accuracy, not all of these digits are output with free-field OUTPUT statements. Instead, the following rules of the free-field convention are used when generating a number's ASCII representation.

ManualsPlus.com

All numbers between 1E−5 and 1E+6 are rounded to 12 significant digits and output in floating-point notation with no leading zeros. If the number is positive, a leading space is output for the sign; if negative, a leading "−" is output.

For example:

```
 32767
-32768
 123456.789012
-.000123456789012
```

If the number is less than 1E−5 or greater than 1E+6, it is rounded to 12 significant digits and output in scientific notation. No leading zeros are output, and the sign character is a space for positive and "−" for negative numbers.

For example:

```
-1.23456789012E+6
 1.23456789012E-5
```

Standard String Format

No leading or trailing spaces are output with the string's characters.

```
String characters.
No leading or trailing spaces.
```

Item Separators and Terminators

Data items are output one byte at a time, beginning with the left-most item in the source list and continuing until all of the source items have been output. Items *in the list* must be *separated* by either a comma or a semicolon. However, items in the data output may or may not be separated by item terminators, depending on the use of item separators in the source lists.

The general sequence of items in the data output is as follows. The end-of-line (EOL) sequence is discussed in the next section.

1st item	item terminator	2nd item	item terminator	. . .	last item	EOL sequence

Using a *comma separator* after an item specifies that the **item terminator** (corresponding to the type of item) will be output after the last character of this item. A carriage-return, CHR$(13), and a line-feed, CHR$(10), terminate string items.

```
OUTPUT Device;"Item",-1234
```

I	t	e	m	CR	LF	−	1	2	3	4	EOL sequence

The default EOL sequence is a CR/LF

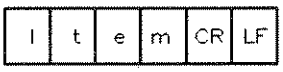
ManualsPlus.com

A comma separator specifies that a comma, CHR$(44), terminates numeric items.

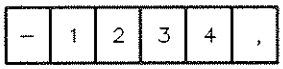
```
OUTPUT Device;-1234,"Item"
```

If a separator follows the last item in the list, the proper item terminator will be output *instead* of the EOL sequence.

```
OUTPUT Device;"Item",
```

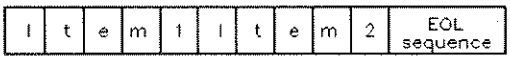



```
OUTPUT Device;-1234,
```



Using a *semicolon separator* suppresses output of the (otherwise automatic) item's terminator.

```
OUTPUT 1;"Item1";"Item2"
```




```
OUTPUT 1;-12;-34
```

If a semicolon separator follows the last item in the list, the EOL sequence and item terminators are suppressed.

```
OUTPUT 1;"Item1";"Item2";
```

Neither of the item terminators nor the EOL sequence are output.

If the item is an array, the separator following the array name determines what is output after each array element. (Individual elements are output in row-major order.)

```
110   DIM Array(1:2,1:3)
120   FOR Row=1 TO 2
130     FOR Column=1 TO 3
140       Array(Row,Column)=Row*10+Column
150     NEXT Column
160   NEXT Row
170   !
180   OUTPUT CRT;Array(*)  ! No trailing separator.
190   !
200   OUTPUT CRT;Array(*), ! Trailing comma.
210   !
220   OUTPUT CRT;Array(*); ! Trailing semi-colon.
230   !
240   OUTPUT CRT;"Done"
250   END
```

ManualsPlus.com

Resultant Output

	1	1	.		1	2	,		1	3	,		2	1	.		2	2	,		2	3	EOL sequence
	1	1	.		1	2	,		1	3	,		2	1	.		2	2	,		2	3	,
	1	1		1	2		1	3		2	1		2	2		2	3							
D	O	N	E	EOL sequence																			

Item separators cause similar action for string arrays.

```
110    DIM Array$(1:2,1:3)[2]
120    FOR Row=1 TO 2
130      FOR Column=1 TO 3
140        Array$(Row,Column)=VAL$(Row*10+Column)
150      NEXT Column
160    NEXT Row
170    !
180    OUTPUT CRT;Array$(*)  ! No trailing separator.
190    !
200    OUTPUT CRT;Array$(*), ! Trailing comma.
210    !
220    OUTPUT CRT;Array$(*); ! Trailing semi-colon.
230    !
240    OUTPUT CRT;"Done"
250    END
```

Resultant Output

1	1	CR	LF	1	2	CR	LF	1	3	CR	LF	2	1	CR	LF	2	2	CR	LF	2	3	EOL sequence	
1	1	CR	LF	1	2	CR	LF	1	3	CR	LF	2	1	CR	LF	2	2	CR	LF	2	3	EOL sequence	
1	1	1	2	1	3	2	1	2	2	2	3												
D	O	N	E	EOL sequence																			

Changing the EOL Sequence

An end-of-line (EOL) sequence is normally sent following the last item sent with OUTPUT. The default EOL sequence consists of a carriage-return and line-feed (CR/LF), sent with no device-dependent END indication. It is also possible to define your own special EOL sequences that include sending special characters, and sending an END indication.

In order to define non-default EOL sequences to be sent by the OUTPUT statement, an I/O path must be used. The EOL sequence is specified in one of the ASSIGN statements that describe the I/O path. An example is as follows.

```
ASSIGN @Device TO 7;EOL CHR$(10)&CHR$(10)&CHR$(13)
```

The characters following EOL are the new EOL-sequence characters. Any character in the range CHR$(0) through CHR$(255) may be included in the string expression that defines the EOL characters; however, the length of the sequence is limited to eight characters or less.

ManualsPlus.com

If END is included in the EOL attribute, an interface-dependent "END" indication is sent with (or after) the last character of the EOL sequence. However, if no EOL sequence is sent, the END indication is also suppressed. The following statement shows an example of defining the EOL sequence to include an END indication.

```
ASSIGN @Device TO 7;EOL CHR$(13)&CHR$(10) END
```

With the HP-IB Interface, the END indication is an End-or-Identify message (EOI) sent with the last EOL character.

The default EOL sequence is a CR and LF sent with no END indication; this default can be restored by assigning EOL OFF to the I/O path.

EOL sequences can also be sent by using the "L" image specifier. See "Outputs that Use Images" for further details.

Using END in Freefield OUTPUT

The secondary keyword END may be optionally specified following the last source-item expression in a freefield OUTPUT statement. The result is to *suppress the End-of-Line (EOL) sequence* that would otherwise be output after the last byte of the last source item. If a comma is used to separate the last item from the END keyword, the corresponding item terminator will be output as before (carriage-return and line-feed for string items and comma for numeric items).

The END keyword has additional significance when the destination is a mass storage file. See the "Data Storage and Retrieval" chapter of *HP Instrument BASIC Programming Techniques* for further details.

Additional Definition

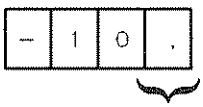
HP Instrument BASIC defines additional action when END is specified in a freefield OUTPUT statement directed to the HP-IB interface.

END with HP-IB Interfaces

With HP-IB interfaces, END has the additional function of sending the End-or-Identify signal (EOI) with the last data byte of the last source item; however, *if no data is sent from the last source item, EOI is not sent.*

Examples

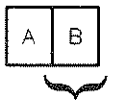
```
ASSIGN @Device TO 701

OUTPUT @Device;-10,END
```



EOI sent with the last character (numeric item terminator).

ManualsPlus.com

```
OUTPUT @Device;"AB";END
```



EOI sent with the last character of the item.

```
OUTPUT @Device;END

OUTPUT @Device;""END
```

Neither EOL sequence nor EOI is sent, since no data is sent.

Outputs that Use Images

The free-field form of the OUTPUT statement is very convenient to use. However, there may be times when the data output by the free-field convention is not compatible with the data required by the receiving device.

Several instances for which you might need to format outputs are: special control characters are to be output; the EOL sequence (carriage-return and line-feed) needs to be suppressed; or the exponent of a number must have only one digit. This section shows you how to use image specifiers to create your own, unique data representations for output operations.

The OUTPUT USING Statement

When this form of the OUTPUT statement is used, the data is output according to the format image referenced by the "USING" secondary keyword. This image consists of one or more individual image specifiers that describe the type and number of data bytes (or words) to be output. The image can be either a string literal, a string variable, or the line label or number of an IMAGE statement. Examples of these four possibilities are listed below.

```
100  OUTPUT 1 USING "6A,SDDD.DDD,3X";" K= ",123.45

100  Image_str$="6A,SDDD.DDD,3X"
110  OUTPUT CRT USING Image_str$;" K= ";123.45

100  OUTPUT CRT USING Image_stmt;" K= ";123.45
110  Image_stmt: IMAGE 6A,SDDD.DDD,3X

100  OUTPUT 1 USING 110;" K= ";123.45
110  IMAGE 6A,SDDD.DDD,3X
```

ManualsPlus.com

Images

Images are used to specify the format of data during I/O operations. Each image consists of groups of individual image (or "field") specifiers, such as 6A, SDDD.DDD, and 3X in the preceding examples. Each of these field specifiers describe one of the following things:

- It describes the desired format of one item in the source list. For example, 6A specifies that a string item is to be output in a "6-character Alpha" field. SDDD.DDD specifies that a numeric item is to be output with Sign, 3 Decimal digits preceding the decimal point, followed by 3 Decimal digits following the decimal point.
- It specifies that special character(s) are to be output. For example, 3X specifies that 3 spaces are to be output. There is no corresponding item in the source list.

Thus, you can think of the image list as either a precise format description or as a procedure. It is convenient to talk about the image list as a procedure for the purpose of explaining how this type of OUTPUT statement is executed.

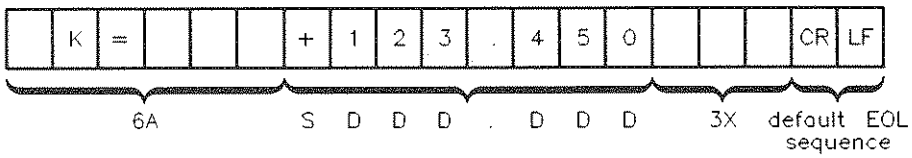
Again, each image list consists of images that each describe the format of data item to be output. The order of images in the list corresponds to the order of data items in the source list. In addition, image specifiers can be added to output (or to suppress the output of) certain characters.

Example of Using an Image

We will use the first of the four, equivalent output statements shown above. Don't worry if you don't understand each of the image specifiers used in the image list; each will be fully described in subsequent sections of this chapter. The main emphasis of this example is that you will see how an image list is used to govern the type and number of characters output.

```
OUTPUT CRT USING "6A,SDDD.DDD,3X";" K= ",123.45
```

The data stream output by the computer is as follows.



Step 1. The computer evaluates the first image in the list. Generally, each group of specifiers separated by commas is an "image"; the commas tell the computer that the image is complete and that it can be "processed". In general, each group of specifiers is processed before going on to the next group. In this case, 6 alphanumeric characters taken from the first item in the source list are to be output.

Step 2. The computer then evaluates the first item in the source list and begins outputting it, one byte (or word) at a time. After the 4th character, the first expression has been "exhausted". In order to satisfy the corresponding specifier, two spaces (alphanumeric "fill" characters) are output.

Step 3. The computer evaluates the next image (note that this image consists of several different image specifiers). The "S" specifier requires that a sign character be

ManualsPlus.com

output for the number, the "D" specifiers require digits of a number, and the "." specifies where the decimal point will be placed. Thus, the number of digits following the decimal point have been specified. All of these specifiers describe the format of the next item in the source list.

Step 4. The next data item in the source list is evaluated. The resultant number is output one digit at a time, according to its image specifiers. A trailing zero has been added to the number to satisfy the "DDD" specifiers following the decimal point.

Step 5. The next image in the list ("3X")is evaluated. This specifier does not "require" data, so the source list needs no corresponding expression. Three spaces are output by this image.

Step 6. Since the entire image list and source list have been "exhausted", the computer then outputs the current (or default, if none has been specified) "end-of-line" sequence of characters (here we assume that a carriage-return and line-feed are the current EOL sequence).

The execution of the statement is now complete. As you can see, the data specified in the source list must match those specified in the output image in type and in number of items.

Image Definitions During Outputs

This section describes the definitions of each of the image specifiers when referenced by OUTPUT statements. The specifiers have been categorized by data type. It is suggested that you scan through the description of each specifier and then look over the examples. You are also highly encouraged to experiment with the use of these concepts.

ManualsPlus.com

Numeric Images

These image specifiers are used to describe the format of numbers.

Sign, Digit, Radix and Exponent Specifiers

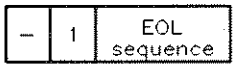
Image Specifier	Meaning
S	Specifies a "+" for positive and a "–" for negative numbers is to be output.
M	Specifies a leading space for positive and a "–" for negative numbers is to be output.
D	Specifies one ASCII digit ("0" through "9") is to to be output. Leading spaces and trailing zeros are used as fill characters. The sign character, if any, "floats" to the immediate left of the most-significant digit. If the number is negative and no S or M is used, one digit specifier will be used for the sign.
Z	Same as "D" except that leading zeros are output. This specifier cannot appear to the right of a radix specifier (decimal point or R).
*	Like D, except that asterisks are output as leading fill characters (instead of spaces). This specifier cannot appear to the right of a radix specifier (decimal point or R).
.	Specifies the position of a decimal point radix-indicator (American radix) within a number. There can be only one radix indicator per numeric image item.
R	Specifies the position of a comma radix indicator (European radix) within a number. There can be only one radix indicator per numeric image item.
E	Specifies that the number is to be output using scientific notation. The "E" must be preceded by at least one digit specifier (D, Z, or *). The default exponent is a four-character sequence consisting of an "E", the exponent sign, and two exponent digits, equivalent to an "ESZZ" image. Since the number is left-justified in the specified digit field, the image for a negative number must contain a sign specifier (see the next section).
ESZ	Same as "E" but only 1 exponent digit is output.
ESZZZ	Same as "E" but three exponent digits are output.
K, –K	Specifies that the number is to be output in a "compact" format, similar to the standard numeric format; however, neither leading spaces (that would otherwise replace a "+" sign) nor item terminators (commas) are output, as would be with the standard numeric format.
H, –H	Like K, except that the number is to be output using a comma radix (European radix).

ManualsPlus.com

Numeric Examples

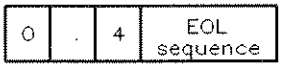
```
OUTPUT @Device USING "DDDD";-123.769
```


```
OUTPUT @Device USING "4D";-1.2
```

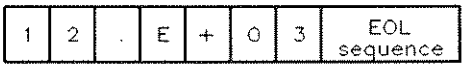



```
OUTPUT @Device USING "ZZ.DD";1.675
```


```
OUTPUT @Device USING "Z.D";.35
```

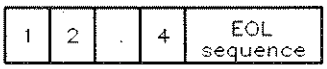



```
OUTPUT @Device USING "DD.E";12345
```

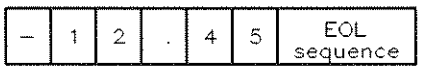



```
OUTPUT @Device USING "2D.DDE";2E-4
```


```
OUTPUT @Device USING "K";12.400
```

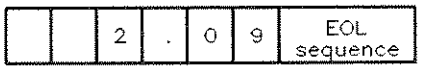



```
OUTPUT CRT USING "MDD.2D";-12.449
```

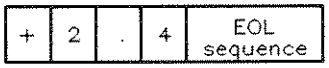


ManualsPlus.com

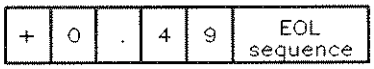
```
OUTPUT CRT USING "MDD.DD";2.09
```




```
OUTPUT 1 USING "SD.D";2.449
```

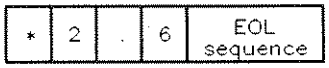



```
OUTPUT 1 USING "SZ.DD";.49
```



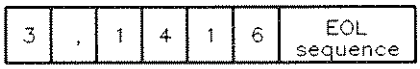

```
OUTPUT CRT USING "SDD.DDE";-2.35
```


```
OUTPUT @Device USING "**.D";2.6
```




```
OUTPUT @Device USING "DRDD";3.1416
```


```
OUTPUT @Device USING "H";3.1416
```



ManualsPlus.com

String Images

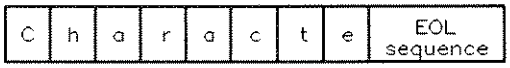
These types of image specifiers are used to specify the format of string data items.

Character Specifiers

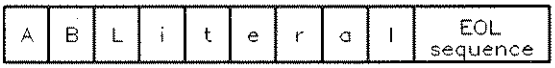
Image Specifier	Meaning
A	Specifies that one character is to be output. Trailing spaces are used as fill characters if the string contains less than the number of characters specified.
"literal"	All characters placed in quotes form a string literal, which is output exactly as is. Literals can be placed in output images, which are part of OUTPUT statements by enclosing them in double quotes.
K, −K, H, −H	Specifies that the string is to be output in "compact" format, similar to the standard string format; however, no item terminators are output as with the standard string format.

String Examples

```
OUTPUT @Device USING "8A";"Characters"
```

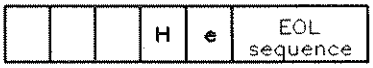



```
OUTPUT @Device USING "K,""Literal""";"AB"
```




```
OUTPUT @Device USING "K";"  Hello  "
```


```
OUTPUT @Device USING "5A";"  Hello  "
```



ManualsPlus.com

Binary Images

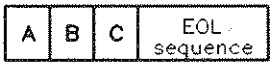
These image specifiers are used to output bytes (8-bit data) and words (16-bit data) to the destination. Typical uses are to output non-ASCII characters or integers in their internal representation.

Binary Specifiers

Image Specifier	Meaning
B	Specifies that one byte (8 bits) of data is to be output. The source expression is evaluated, rounded to an integer, and interpreted MOD 256. If it is less than −32 768, CHR$(0) is output. If is greater than 32 767, CHR$(255) is output.
W	Specifies that one word of data (16 bits) are to be sent as a 16-bit, two's-complement integer. The corresponding source expression is evaluated and rounded to an integer. If it is less than −32 768, then −32 768 is sent; if it is greater than 32 767, then 32 767 is sent. If the destination is a BDAT or HPUX file, or string variable, the WORD attribute is ignored and all data are sent as bytes; however, pad byte(s), CHR$(0), will also be output whenever necessary to achieve alignment on a word boundary. Since HP Instrument BASIC only supports 8-bit interfaces, two bytes are always output, with the most significant byte first. This image specifier has been included primarily to maintain compatibility with HP Series 200/300 BASIC programs that include this specifier.
Y	Like W, except that no pad bytes are output to achieve alignment on a word boundary.

Binary Examples

```
OUTPUT @Device USING "B,B,B";65,66,67
```




```
OUTPUT @Device USING "B";13
```


```
OUTPUT @Device USING "W";256*65+66
```

ManualsPlus.com

Special-Character Images

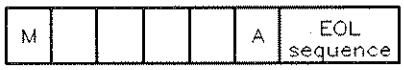
These specifiers require no corresponding data in the source list. They can be used to output spaces, end-of-line sequences, and form-feed characters.

Special-Character Specifiers

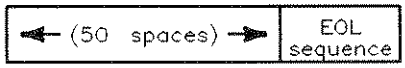
Image Specifier	Meaning
X	Specifies that a space character, CHR$(32), is to be output.
/	Specifies that a carriage-return character, CHR$(13), and a line-feed character, CHR$(10), are to be output.
@	Specifies that a form-feed character, CHR$(12), is to be output.

Special-Character Examples

```
OUTPUT @Device USING "A,4X,A";"M","A"
```

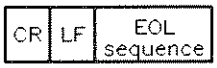



```
OUTPUT @Device USING "50X"
```




```
OUTPUT @Device USING "@,/"
```


```
OUTPUT @Device USING "/"
```



ManualsPlus.com

Termination Images

These specifiers are used to output or suppress the end-of-line sequence output after the last data item.

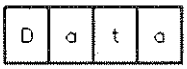
Termination Specifiers

Image Specifier	Meaning
L	Specifies that the current end-of-line sequence is to be output. The default EOL characters are CR and LF; see "Changing the EOL Sequence" for details on how to redefine these characters.
#	Specifies that the EOL sequence that normally follows the last item is to be suppressed.
%	Is ignored in output images but is allowed to be compatible with ENTER images.
+	Specifies that the EOL sequence that normally follows the last item is to be replaced by a single carriage-return character (CR).
–	Specifies that the EOL sequence that normally follows the last item is to be replaced by a single line-feed character (LF).

Termination Examples

```
OUTPUT @Device USING "4A,L";"Data"
```


```
OUTPUT @Device USING "#,K";"Data"
```




```
OUTPUT @Device USING "#,B";12
```


```
OUTPUT @Device USING "+,K";"Data"
```

ManualsPlus.com

```
OUTPUT @Device USING "-,L,K";"Data"
```

Additional Image Features

Several additional features of outputs that use images are available with the computer. Several of these features, which have already been shown, will be explained here in detail.

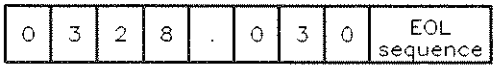
Repeat Factors

Many of the specifiers can be repeated without having to explicitly list the specifier as many times as it is to be repeated. For instance, to a character field of 15 characters, you do not need to use "AAAAAAAAAAAAAAA"; instead, you merely specify the number of times that the specifier is to be repeated in front of the image ("15A"). The following specifiers can be repeated by specifying an integer repeat factor; the specifiers not listed cannot be repeated in this manner.

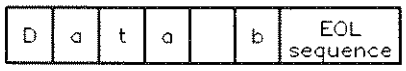
Repeatable Specifiers	Nonrepeatable Specifiers
Z, D, A, X, /, @, L	S, M, ., R, E, K, H, B, W, Y, #, %, +, -

Examples

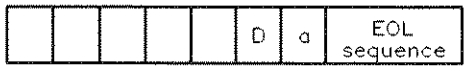
```
OUTPUT @Device USING "4Z.3D";328.03
```




```
OUTPUT @Device USING "6A";"Data bytes"
```

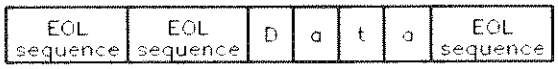



```
OUTPUT @Device USING "5X,2A";"Data"
```



ManualsPlus.com

```
OUTPUT @Device USING "2L,4A";"Data"
```




```
OUTPUT @Device USING "8A,2@";"The End"
```


```
OUTPUT @Device USING "2/"
```

Image Re-Use

If the number of items in the source list exceeds the number of matching specifiers in the image list, the computer attempts to reuse the image(s) beginning with the first image.

```
110   ASSIGN @Device TO CRT
120   Num_1=1
130   Num_2=2
140   !
150   OUTPUT @Device USING "K";Num_1,"Data_1",Num_2,"Data_2"
160   OUTPUT @Device USING "K,/";Num_1,"Data_1",Num_2,"Data_2"
170   END
```

Resultant Display

```
1Data_12Data_2
1
Data_1
2
Data_2
```

Since the "K" specifier can be used with both numeric and string data, the above OUTPUT statements can reuse the image list for all items in the source list. If any item cannot be output using the corresponding image item, an error results. In the following example, "Error 100 in 150" occurs due to data mismatch.

```
110   ASSIGN @Device TO CRT
120   Num_1=1
130   Num_2=2
140   !
150   OUTPUT @Device USING "DD.DD";Num_1,Num_2,"Data_1"
160   END
```

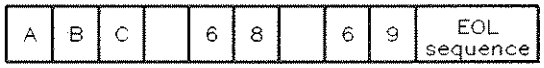
ManualsPlus.com

Nested Images

Another convenient capability of images is that they can be nested within parentheses. The entire image list within the parentheses will be used the number of times specified by the repeat factor preceding the first parenthesis. The following program is an example of this feature.

```
100   ASSIGN @Device TO 701
110   !
120   OUTPUT @Device USING "3(B),X,DD,X,DD";65,66,67,68,69
130   END
```

Resultant Output



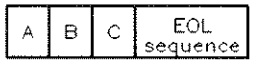
This nesting with parentheses is made with the same hierarchy as with parenthetical nesting within mathematical expressions. Only eight levels of nesting are allowed.

END with OUTPUTs that Use Images

Using the optional secondary keyword END in an OUTPUT statement that uses an image produces results that differ from those of using END in a freefield OUTPUT statement. Instead of always suppressing the EOL sequence, the END keyword *only suppresses the EOL sequence when no data are output from the last source-list expression*. Thus, the "#" image specifier generally controls the suppression of the otherwise automatic EOL sequence, while the END keyword suppresses it only in less common usages.

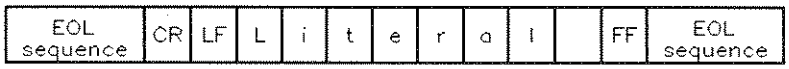
Examples

```
Device=12

OUTPUT Device USING "K";"ABC",END
OUTPUT Device USING "K";"ABC";END
OUTPUT Device USING "K";"ABC" END
```



The EOL sequence is not suppressed.

```
OUTPUT Device USING "L,/,""Literal"",X,@"
```



ManualsPlus.com

In this case, specifiers that require no source-item expressions are used to generate characters for the output; there are no source expressions. The EOL sequence is output after all specifiers have been used to output their respective characters. Compare this action to that shown in the next example.

```
OUTPUT Device USING "L,/,""Literal"",X,@";END
```

The EOL sequence is suppressed because no source items were included in the statement; all characters output were the result of specifiers that require no corresponding expression in the source list.

Additional END Definition

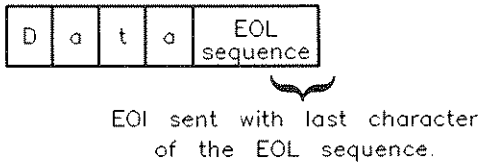
The END secondary keyword has been defined to produce additional action when included in an OUTPUT statement directed to HP-IB interfaces.

END with HP-IB Interfaces

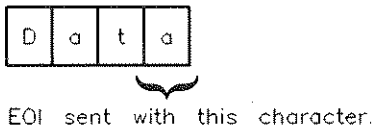
With HP-IB interfaces, END has the additional function of sending the End-or-Identify signal (EOI) with the *last character* of either the last source item or the EOL sequence (if sent). As with freefield OUTPUT, *no EOI is sent if no data is sent from the last source item and the EOL sequence is suppressed.*

Examples.

```
ASSIGN @Device TO 701

OUTPUT @Device USING "K";"Data",END
OUTPUT @Device USING "K";"Data","",END
```



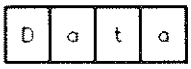

```
OUTPUT @Device USING "#,K";"Data" END
```



EOI is sent with the last character of the last source item when the EOL sequence is suppressed, because the last source item contained data that was used in the output.

```
OUTPUT @Device USING "#,K";"Data","",END
OUTPUT @Device USING """Data""";END
```

ManualsPlus.com



The EOI was not sent in either case, since no data were sent from the last source item *and* the EOL sequence was suppressed.

ManualsPlus.com

5

Entering Data

This chapter discusses the topic of entering data from devices. You may already be familiar with the OUTPUT statement described in the previous chapter; many of those concepts are applicable to the process of entering data. Earlier in this manual, you were told that *the data output from the sender had to match that expected by the receiver.* Because of the many ways that data is represented in external devices, entering data can sometimes require more programming skill than outputting data. In this chapter, you will see what is involved in being the receiving device. Both free-field enters and enters that use images are described, and several examples are given with each topic.

Free-Field Enters

Executing the free-field form of the ENTER invokes conventions that are the "converse" of those used with the free-field OUTPUT statement. In other words, data output using the free-field form of the OUTPUT statement can be readily entered using the free-field ENTER statement; no explicit image specifiers are required. The following statements exemplify this form of the ENTER statement.

For example:

```
ENTER @Voltmeter;Reading
ENTER 724;Readings(*)
ENTER From_string$;Average,Student_name$
ENTER @From_file;Data_code,Str_element$(X,Y)
```

Item Separators

Destination items in ENTER statements can be separated by *either* a comma or a semicolon. Unlike the OUTPUT statement, it makes *no difference* which is used; data will be entered into each destination item in a manner independent of the punctuation separating the variables in the list. However, *no trailing punctuation is allowed*. The first two of the following statements are equivalent, but an error is reported when the third statement is executed.

For example:

```
ENTER @From_a_device;N1,N2,N3
ENTER @From_a_device;N1;N2;N3
```

ManualsPlus.com

Item Terminators

Unless the receiver knows exactly how many characters are to be sent, each data item output by the sender must be terminated by special character(s). When entering ASCII data with the free-field form of the ENTER statement, the computer does not know how many characters will be output by the sender.

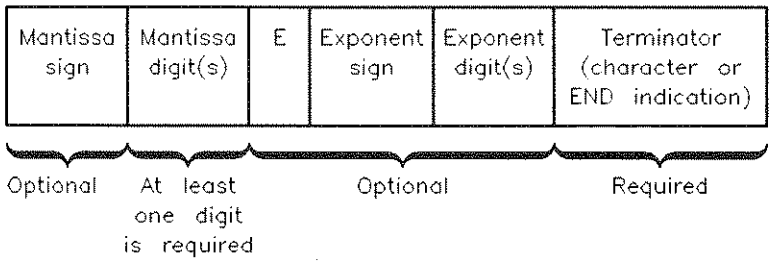
Item terminators must signal the end of each item so that the computer enters data into the proper destination variable. The terminator of the last item may also terminate the ENTER statement (in some cases). The actual character(s) that terminate entry into each type of variable are described in the next sections.

In addition to the termination characters, each item can be terminated (only with selected interfaces) by a device-dependent END indication. For instance, some interfaces use a signal known as EOI (End-or-Identify). The EOI signal is only available with the HP-IB, and keyboard interfaces. EOI termination is further described in the next sections.

Entering Numeric Data with the Number Builder

When the free-field form of the ENTER statement is used, numbers are entered by a routine known as the "number builder". This firmware routine evaluates the incoming ASCII numeric characters and then "builds" the appropriate internal-representation number. This number builder routine recognizes whether data being entered is to be placed into an INTEGER or REAL variable and then generates the appropriate internal representation.

The number builder is designed to be able to enter several formats of numeric data. However, the general format of numeric data must be as follows to be interpreted properly by HP Instrument BASIC.



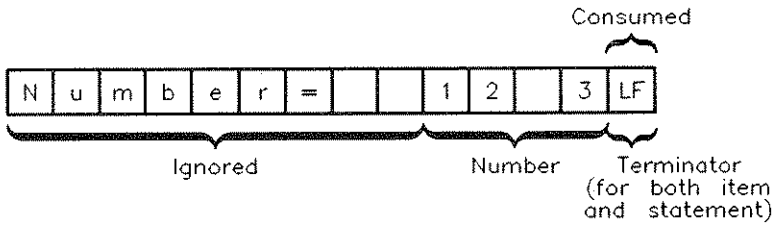
Numeric characters include decimal digits "0" through "9" and the characters ".", "+", "−", "E", and "e". These last five characters must occur in meaningful positions in the data stream to be considered numeric characters; if any of them occurs in a position in which it cannot be considered part of the number, it will be treated as a non-numeric character.

ManualsPlus.com

The following rules are used by the number builder to construct numbers from incoming streams of ASCII numeric characters.

1. All leading non-numerics are ignored; all leading and embedded spaces are ignored.

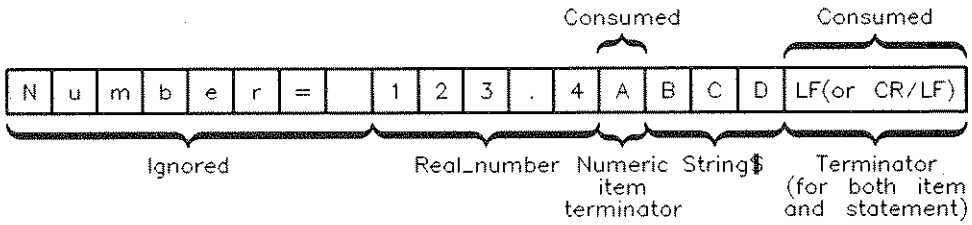
```
100   ASSIGN @Device TO Device_selector
110   ENTER @Device;Number  ! Default is data type REAL.
120   END
```



The result of entering the preceding data with the given ENTER statement is that Number receives a value of 123. The line-feed (statement terminator) is *required* since Number is the last item in the destination list.

2. Trailing non-numerics terminate entry into a numeric variable, and the terminating characters (of *both* string and numeric items) are "consumed". In this manual, "consumed" characters refers to characters *used to terminate* an item but not entered into the variable; "ignored" characters are entered but are *not used*.

```
ENTER @Device;Real_number,String$
```

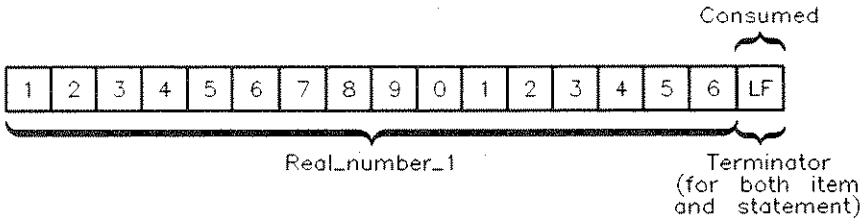


The result of entering the preceding data with the given ENTER statement is that Real_number receives the value 123.4 and String$ receives the characters "BCD". The "A" was lost when it terminated the numeric item; the string-item terminator(s) are also lost. The string-item terminator(s) also terminate the ENTER statement, since String$ is the last item in the destination list.

3. If more than 16 digits are received, only the first 16 are used as significant digits. However, all additional digits are treated as trailing zeros so that the exponent is built correctly.

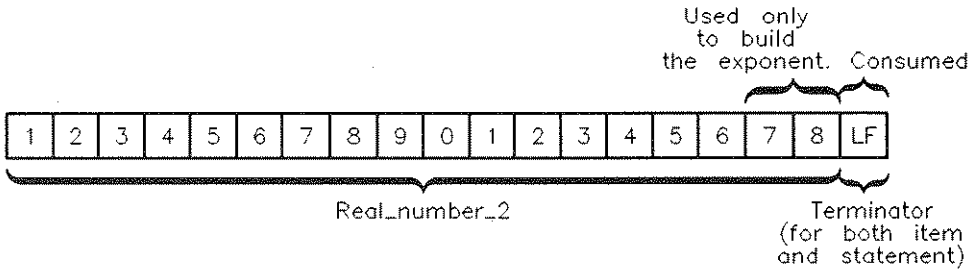
ManualsPlus.com

```
ENTER @Device;Real_number_1
```



The result of entering the preceding data with the given ENTER statement is that `Real_number_1` receives the value `1.234567890123456 E+15`.

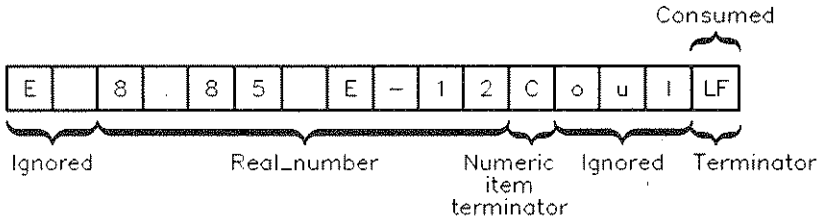
```
ENTER @Device;Real_number_2
```



The result of entering the preceding data with the given ENTER statement is that `Real_number_2` receives the value `1.234567890123456 E+17`.

4. Any exponent sent by the source must be preceded by at least one mantissa digit *and* an `E`(or `e`) character. If no exponent digits follow the `E` (or `e`), no exponent is recognized, but the number is built accordingly.

```
ENTER @Device;Real_number
```

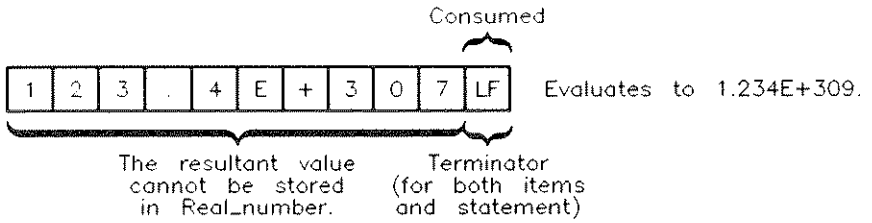


The result of entering the preceding data with the given ENTER statement is that Real_number receives a value of 8.85 E−12. The character "C" terminates entry into Real_number, and the characters "oul" are entered (but ignored) in search of the required line-feed statement terminator. If the character "C" is to be entered but not ignored, you must use an image. Using images with the ENTER statement is described later in this chapter.

ManualsPlus.com

5. If a number evaluates to a value outside the range corresponding to the type of the numeric variable, an error is reported. If no type has been declared explicitly for the numeric variable, it is assumed to be REAL.

```
ENTER @Device;Real_number
```



The data is entered but evaluates to a number outside the range of REAL numbers. Consequently, error 19 is reported, and the variable `Real_number` retains its former value.

6. If the item is the *last* one in the list, *both* the *item* and the *statement* need to be properly *terminated*. If the numeric **item** is terminated by a non-numeric character, the **statement** will *not* be terminated until it either receives a line-feed character or an END indication (such as EOI signal with a character). The topic of terminating free-field ENTER statements is described later.

Entering String Data

Strings are groups of ASCII characters of varying lengths. Unlike numbers, almost any character can appear in any position within a string; there is not really any defined structure of string data. The routine used to enter string data is therefore much simpler than the number builder. It only needs to keep track of the dimensioned length of the string variable and look for string-item terminators (such as CR/LF, LF, or EOI sent with a character).

String-item terminator characters are either a line-feed (LF) or a carriage-return followed by a line-feed (CR/LF). As with numeric-item terminators characters, these characters are not entered into the string variable (during free-field enters); they are "lost" when they terminate the entry. The EOI signal also terminates entry into a string variable, but the variable must be the last item in the destination list (during free-field enters).

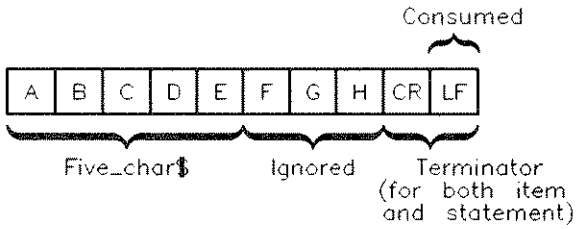
All characters received from the source are entered directly iemph appropriate string variable until *any* of the following conditions occurs:

- An item terminator character is received.
- The number of characters entered equals the dimensioned length of the string variable.
- The EOI signal is received.

The following statements and resultant variable contents illustrate the first two conditions; the next section describes termination by EOI. Assume that the string variables Five_char$ and Ten_char$ are dimensioned to lengths of 5 and 10 characters, respectively.

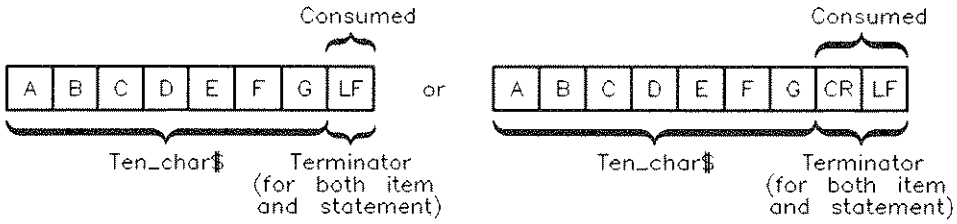
ManualsPlus.com

```
ENTER @Device;Five_char$
```



The variable Five_char$ only receives the characters "ABCDE", but the characters "FGH" are entered (and ignored) in search of the terminating carriage-return/line-feed (or line-feed).

```
ENTER @Device;Ten_char$
```



The result of entering the preceding data with the given ENTER statement is that Ten_char$ receives the characters "ABCDEFG" and the terminating LF (or CR/LF) is lost.

ManualsPlus.com

Terminating Free-Field ENTER Statements

Terminating conditions for free-field ENTER statements are as follows.

1. If the *last item* is terminated by a line-feed or by a character accompanied by EOI, the *entire statement* is properly terminated.
2. If an *END indication* is received while entering data into the *last item*, the statement is properly terminated. Examples of END indications are encountering the last character of a string variable while entering data from the variable and receiving EOI with a character.
3. If one of the preceding *statement-termination* conditions has *not* occurred *but* entry into the *last item* has been terminated,up to 256 *additional* characters are entered in search of a termination condition. If one is not found, an error occurs.

One case in which this termination condition may not be obvious can occur while entering string data. If the last variable in the destination list is a string *and* the dimensioned length string has been reached *before* a terminator is received, additional characters are entered (but ignored) until the terminator is found. The reason for this action is that the next characters received are still part of this data item, as far as the data *sender* is concerned. These characters are accepted from the sender so that the next enter operation will not receive these "leftover" characters.

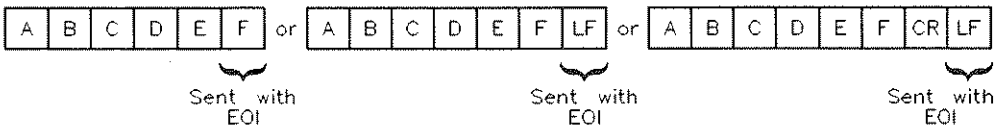
Another case involving numeric data can also occur. (See the example given with "rule 4" describing the number builder.) If a trailing non-numeric character terminates the last item (which is a numeric variable), additional characters will be entered in search of either a line-feed or a character accompanied by EOI. Unless this terminating condition is found before 256 characters have been entered, an error is reported.

EOI Termination

A termination condition for the HP-IB Interface is the EOI (End-or-Identify) signal. When this message is sent, it immediately terminates the entire ENTER statement, regardless of whether or not all variables have been satisfied. However, if all variable items in the destination list have not been satisfied, an error is reported.

For example:

```
ENTER @Device;String$
```

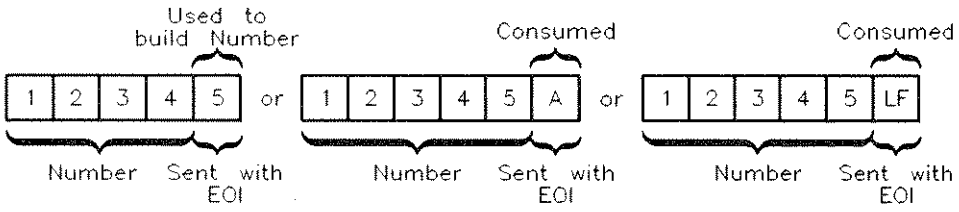


The result of entering the preceding data with the given ENTER statement is that String$ receives the characters "ABCDEF". The EOI signal being received with either the last character or with the terminator character properly terminates the ENTER statement. If the character accompanied by EOI is a string character (not a terminator), it is entered into the variable as usual.

ManualsPlus.com

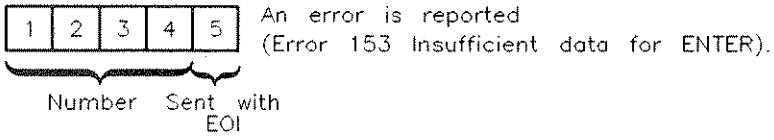
For example:

```
ENTER @Device;Number
```



The result of entering any of the above data streams with the given ENTER statement is that Number receives the value 12345. If the EOI signal accompanies a numeric character, it is entered and used to build the number; if the EOI is received with a numeric terminator, the terminator is lost as usual.

```
ENTER @Device;Number,String$
```



The result of entering the preceding data with the given statement is that an error is reported when the character "5" accompanied by EOI is received. However, Number receives the value 12345, but String$ retains its previous value. An error is reported because *all* variables in the destination list have *not* been satisfied when the EOI is received. Thus, the EOI signal is an *immediate statement terminator during free-field enters*. The EOI signal has a *different* definition during enters that use images, as described later in this chapter.

Enters that Use Images

The free-field form of the ENTER statement is very convenient to use; the computer automatically takes care of placing each character into the proper destination item. However, there are times when you need to design your own images that match the format of the data output by sources. Several instances for which you may need to use this type of enter operations are: the incoming data does not contain any terminators; the data stream is not followed by an end-of-line sequence; or two consecutive bytes of data are to be entered and interpreted as a two's-complement integer.

ManualsPlus.com

The ENTER USING Statement

The means by which you can specify how the computer will interpret the incoming data is to reference an image in the ENTER statement. The four general ways to reference the image in ENTER statements are as follows.

```
100  ENTER @Device_x USING "6A,DDD.DD";String_var$,Num_var

100  Image_str$="6A,DDD.DD"
110  ENTER @Device_x USING Image_str$;String_var$,Num_var

100  ENTER @Device USING Image_stmt;String_var$,Num_var
110  Image_stmt: IMAGE 6A,DDD.DD

100  ENTER @Device USING 110;String_var$,Num_var
110  IMAGE 6A,DDD.DD
```

Images

Images are used to specify how data entered from the source is to be interpreted and placed into variables; each image consists of one or more groups of individual image specifiers that determine how the computer will interpret the incoming data bytes (or words). Thus, image lists can be thought of as a description of *either*

- the format of the expected data, or
- the procedure that the ENTER statement will use to enter and interpret the incoming data bytes.

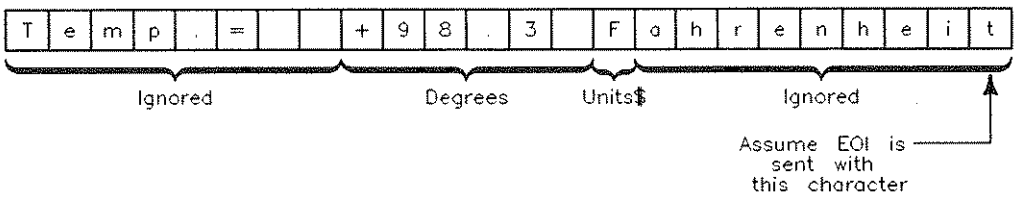
The examples given here treat the image list as a *procedure*.

All of the image specifiers used in image lists are valid for both enters and outputs. However, most of the specifiers have a slightly different meaning for each operation. If you plan to use the same image for output and enter, you must fully understand how both statements will use the image.

Example of an Enter Using an Image

This example is used to show you exactly how the computer uses the image to enter incoming data into variables. Look through the example to get a general feel for how these enter operations work. Afterwards, you should read the descriptions of the pertinent specifier(s).

Assume that the following stream of data bytes are to be entered into the computer.



ManualsPlus.com

Given the preceding conditions, let's look at how the computer executes the following ENTER statement that uses the specified IMAGE statement.

```
300  ENTER @Device USING Image_1;Degrees,Units$
310  Image_1:  IMAGE 8X,SDDD.D,A
```

Step 1. The computer evaluates the first image of the IMAGE statement. It is a special image in that it does not correspond to a variable in the destination list. It specifies that eight characters of the incoming data stream are to be ignored. Eight characters, "Temp.= ", are entered and are ignored (i.e., are not entered into any variable).

Step 2. The computer evaluates the next image. It specifies that the next six characters are to be used to build a number. Even though the order of the sign, digit, and radix are explicitly stated in the image, the actual order of these characters in the incoming data stream does not have to match this specifier exactly. Only the *number* of numeric specifiers in the image (here, six) is all that is used to specify the data format. When all six characters have been entered, the number builder attempts to form a number.

Step 3. After the number is built, it is placed into the variable "Degrees"; the representation of the resultant number depends on the numeric variable's type (INTEGER, or REAL).

Step 4. The next image in the IMAGE statement is evaluated. It requires that one character be entered for the purpose of filling the variable "Units$". One byte is then entered into Units$.

Step 5. All images have been satisfied; however, the computer has not yet detected a statement-terminating condition. A line-feed or a character accompanied by EOI must be received to terminate the ENTER statement. Characters are then entered, but ignored, in search of one of these conditions. The statement is terminated when the EOI is sent with the "t". For further explanation, see "Terminating Enters that Use Images".

The above example should help you to understand how images are used to determine the interpretation of incoming data. The next section will help you to use each specifier to create your desired images.

Image Definitions During Enter

This section describes the individual image specifiers in detail. The specifiers have been categorized into data and function type.

ManualsPlus.com

Numeric Images

Sign, digit, radix, and exponent specifiers are all used identically in ENTER images. The number builder can also be used to enter numeric data.

Numeric Specifiers

Image Specifier	Meaning
D	Specifies that one byte is to be entered and interpreted as a numeric character. If the characters is non-numeric (including leading spaces and item terminators), it will still "consume" one digit of the image item.
Z, *	Same action as D. Keep in mind that A and * can only appear to the left of the radix indicator (decimal point or R) in a numeric image item.
S, M	Same action as D in that one byte is to be entered and interpreted as a numeric character. At least one digit specifier must follow either of these specifiers in an image item.
.	Same action as D in that one byte is to be entered and interpreted as a numeric character. At least one digit specifier must accompany this specifier in an image item.
R	Same action as D in that one byte is to be entered and interpreted as a numeric character; however, when R is used in a numeric image, it directs the number builder to use the comma as a radix indicator and the period as a terminator to the numeric item. At least one digit specifier must accompany this specifier in the image item.
E	Equivalent to 4D, if preceded by at least one digit specifier (Z, *, or D) in the image item.
	The following specifiers must also be preceded by at least one digit specifier.
ESZ	Equivalent to 3D.
ESZZ	Equivalent to 4D.
ESZZZ	Equivalent to 5D.
K, −K	Specifies that a variable number of characters are to be entered and interpreted according to the rules of the number builder (same rules as used in "free-field" ENTER operations).
H, −H	Like K, except that a comma is used as the radix indicator, and a period is used as the terminator for the numeric item.

Examples of Numeric Images

These 5 are equivalent:

```
ENTER @Device USING "SDD.D";Number
ENTER @Device USING "3D.D";Number
ENTER @Device USING "5D";Number
ENTER @Device USING "DESZZ";Number
ENTER @Device USING "**.DD";Number
```

Use the rules of the number builder:

```
ENTER Device USING "K";Number
```

ManualsPlus.com

Enter five characters, using comma as radix:

```
ENTER @Device USING "DDRDD";Number
```

Use the rules of the number builder, but use the comma as radix:

```
ENTER @Device USING "H";Number
```

String Images

The following specifiers are used to determine the number of and the interpretation of data bytes entered into string variables.

String Specifiers

Image Specifier	Meaning
A	Specifies that one byte is to be entered and interpreted as a string character. Any terminators are entered into the string when this specifier is used.
K, H	Specifies that "free-field" ENTER conventions are to be used to enter data into a string variable; characters are entered directly into the variable until a terminating condition is sensed (such as CR/LF, LF, or an END indication).
−K, −H	Like K, except that line-feeds (LF's) do not terminate entry into the string; instead, they are treated as string characters and placed in the variable. Receiving an END indication terminates the image item (for instance, receiving EOI with a character on an HP-IB interface, encountering an end-of-data, or reaching the variable's dimensioned length).
L, @	These specifiers are ignored for ENTER operations; however, they are allowed for compatibility with OUTPUT statements (that is, so that one image may be used for *both* ENTER and OUTPUT statements). Note that it may be necessary to skip characters (with specifiers such as X or /) when ENTERing data that has been sent by including these specifiers in an OUTPUT statement.

Examples of String Images

Enter 10 characters:

```
ENTER @Device USING "10A";Ten_chars$
```

Enter using the free-field rules:

```
ENTER @Device USING "K";Any_string$
```

Enter two strings:

```
ENTER @Device USING "5A,K";String$,Number$
```

Enter a string and a number:

```
ENTER @Device USING "5A,K";String$,Number
```

Enter characters until string is full or END is received:

```
ENTER @Device USING "-K";All_chars$
```

ManualsPlus.com

Ignoring Characters

These specifiers are used when one or more characters are to be ignored (i.e., entered but not placed into a string variable).

Specifiers Used to Ignore Characters

Image Specifier	Meaning
X	Specifies that a character is to be entered but ignored (not placed into a variable).
"literal"	Specifies that the number of characters in the literal are to be entered but ignored (not placed into a variable).
/	Specifies that all characters are to be entered but ignored (not placed into a variable) until a line-feed is received. EOI is also ignored until the line-feed is received.

Examples of Ignoring Characters

Ignore first five and use second five characters:

```
ENTER @Device USING "5X,5A";Five_chars$
```

Ignore 6th through 9th characters:

```
ENTER @Device USING "5A,4X,10A";S_1$,S_2$
```

Ignore 1st item of unknown length:

```
ENTER @Device USING "/,K";String2$
```

Ignore two characters:

```
ENTER @Device USING """zz"",AA";S_2$
```

Binary Images

These specifiers are used to enter one byte (or word) that will be interpreted as a number.

Binary Specifiers

Image Specifier	Meaning
B	Specifies that one byte is to be entered and interpreted as an integer in the range 0 through 255.
W	Specifies that one 16-bit word is to be entered and interpreted as a 16-bit, two's complement INTEGER. Since all HP Instrument BASIC interfaces are 8-bit, two bytes are always entered; the first byte entered is most significant. If the source is a file, or string variable, all data are entered as bytes; however, one byte may still be entered and ignored when necessary to achieve alignment on a word boundary.
Y	Like W, except that pad bytes are never entered to achieve word alignment.

ManualsPlus.com

Examples of Binary Images

Enter three bytes, then look for LF or END indication:

```
ENTER @Device USING "B,B,B";N1,N2,N3
```

Enter the first two bytes as an INTEGER, then the rest as string data:

```
ENTER @Device USING "W,K";N,N$
```

Terminating Enters that Use Images

This section describes the default statement-termination conditions for enters that use images (for devices). The effects of numeric-item and string-item terminators and the end-or-identify (EOI) signal during these operations are discussed in this section. After reading this section, you will be able to better understand how enters that use images work and how the default statement-termination conditions are *modified* by the **#**, **%**, **+**, and **-** image specifiers.

Default Termination Conditions

The default statement-termination conditions for enters that use images are very similar to those required to terminate free-field enters. *Either* of the following conditions will properly terminate an ENTER statement that uses an image.

- An END indication (such as the EOI signal or end-of-data) is received *with* the byte that satisfies the last *image item within 256 bytes after* the byte that satisfied the last image item.
- A line-feed is received *as* the byte that satisfies the last *image item* (exceptions are the "B" and "W" specifiers) or *within 256 bytes after* the byte that satisfied the last image item.

EOI Redefinition

It is important to realize that when an enter uses an image (when the secondary keyword "USING" is specified), the definition of the EOI signal is *automatically modified*. If the EOI signal terminates the *last image item*, the entire statement is properly terminated, as with free-field enters. In addition, *multiple EOI signals are now allowed* and act as *item* terminators; however, the EOI must be received *with* the byte that satisfies each image item. If the EOI is received *before* any image is satisfied, it is *ignored*. Thus, all images must be satisfied, and EOI will not cause early termination of the ENTER-USING-image statement.

The following table summarizes the definitions of EOI during several types of ENTER statement. The statement-terminator modifiers are more fully described in the next section.

ManualsPlus.com

Effects of EOI During ENTER Statements

	Free-Field ENTER Statements	ENTER USING without # or %	ENTER USING with #	ENTER USING with %
Definition of EOI	Immediate statement terminator	Item terminator or statement terminator	Item terminator or statement terminator	Immediate statement terminator
Statement Terminator Required?	Yes	Yes	No	No
Early Termination Allowed?	No	No	No	Yes

Statement-Termination Modifiers

These specifiers modify the conditions that terminate enters that use images. The first one of these specifiers encountered in the image list modifies the termination conditions for the ENTER statement. If another of these specifiers is encountered in the image list, it again modifies the terminating conditions for the statement.

Statement-Termination Modifiers

Image Specifier	Meaning
#	Specifies that a statement-termination condition is *not* required; the ENTER statement is automatically terminated as soon as the *last image item* is satisfied.
%	Also specifies that a statement-termination condition is not required. In addition, EOI is redefined to be an *immediate* statement terminator, *allowing early termination* of the ENTER *before all* image items have been satisfied. However, the statement can only be terminated on a "legal item boundary". The legal boundaries for different specifiers are as follows: **Specifier** — **Legal Boundary** K,−K — With any character, since this specifies a variable-width field of characters. S,M,D,E,Z,.,A,X,*literal*,B,W — Only with the last character that satisfies the image (e.g., with the 5th character of a `5A` image). If EOI is received with any other character, it is ignored. / — Only with the last line-feed character that satisfies the image (e.g., with the 3rd line-feed of a "3/" image); otherwise, it is ignored.
+	Specifies that an END indication is required to terminate the ENTER statement. Line-feeds are ignored as statement terminators; however, they will still terminate items (unless a −K or −H image is used for strings).
−	Specifies that a line-feed is required to terminate the statement. EOI is ignored, and other END indications (such as EOF or end-of-data) cause an error if encountered before the line-feed.

ManualsPlus.com

Examples of Modifying Termination Conditions

Enter a single byte:

```
ENTER @Device USING "#,B";Byte
```

Enter a single word:

```
ENTER @Device USING "#,W";Integer
```

Enter an array, allowing early termination by EOI:

```
ENTER @Device USING ",K";Array(*)
```

Enter characters into `String$` until line-feed received, then continue entering characters it until END received:

```
ENTER @Device USING "+,K";String$
```

Enter characters until line-feed received; ignore EOI, if received:

```
ENTER @Device USING "-,K";String$
```

Additional Image Features

Several additional image features are available with this BASIC language. Some of these features have already been shown in examples, and all of them resemble the additional features of images used with OUTPUT statements.

Repeat Factors

All of the following specifiers can be preceded by an integer that specifies how many times the specifier is to be used.

Repeatable Specifiers	Non-Repeatable Specifiers
Z, D, A, X, /, @, L	S, M, ., R, E, K, H, B, W, Y, #, %, +, -

Image Reuse

If there are fewer images than items in the destination list, the list will be reused, beginning with the first item in the image list. If there are more images than there are items, the additional specifiers will be ignored.

Examples

The "B" is reused:

```
ENTER @Device USING "#,B";B1,B2,B3
```

The "W" is not used:

```
ENTER @Device USING "2A,2A,W";A$,B$
```

ManualsPlus.com

Nested Images

Parentheses can be used to nest images within the image list. The hierarchy is the same as with mathematical operations; evaluation is from inner to outer sets of parentheses. The maximum number of levels of nesting is eight.

Example

```
ENTER @Source USING "2(B,5A,/),/";N1,N1$,N2,N2$
```

ManualsPlus.com

ManualsPlus.com

6

I/O Path Attributes

This chapter contains two major topics, both of which involve additional features provided by I/O path names.

- The first topic is that I/O path names can be given attributes which control the way that the system handles the data sent and received through the I/O path. Attributes are available for such purposes as controlling data representations and defining special end-of-line (EOL) sequences.
- The second topic is that one set of I/O statements can access most system resources instead of using a separate set of statements to access each class of resources. This second topic, herein called "unified I/O", may be considered an implicit attribute of I/O path names.

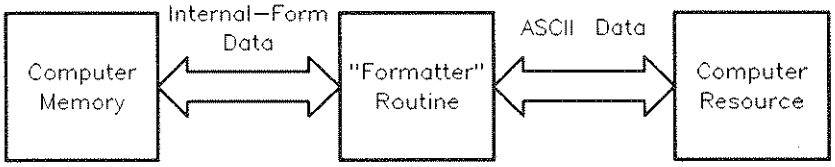
The FORMAT Attributes

All I/O paths possess one of the two following attributes:

- FORMAT ON—means that the data are sent in ASCII representation.
- FORMAT OFF—means that the data are sent in BASIC internal representation.

Before getting into how to assign these attributes to I/O paths, let's take a brief look at each one.

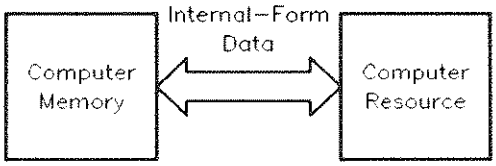
With FORMAT ON, internally represented numeric data must be "formatted" into its ASCII representation before being sent to the device. Conversely, numeric data being received from the device must be "unformatted" back into its internal representation. These operations are shown in the diagrams below:



Numeric Data Transformations with FORMAT ON

With FORMAT OFF, however, no formatting is required. The data items are merely copied from the source to the destination. This type of I/O operation requires less time, since fewer steps are involved.

ManualsPlus.com



Numeric Data Transfer with FORMAT OFF

The only requirement is that the resource also use the exact same data representations as the internal HP Instrument BASIC representation.

Here are how each type of data item is represented and sent with FORMAT OFF:

- INTEGER: two-byte (16-bit), two's complement.
- REAL: eight-byte (64-bit) IEEE floating-point standard.
- String: four-byte (32-bit) length header, followed by ASCII characters. An additional ASCII space character, CHR$(32), may be sent and received with strings in order to have an even number of bytes.

Here are the FORMAT OFF rules for OUTPUT and ENTER operations:

- No item terminator and no EOL sequence are sent by OUTPUT.
- No item terminator and no statement-termination conditions are required by ENTER.
- If either OUTPUT or ENTER uses an IMAGE (such as with OUTPUT 701 USING "4D.D"), then the FORMAT ON attribute is *automatically* used.

Assigning Default FORMAT Attributes

As discussed in the "Directing Data Flow" chapter, names are assigned to I/O paths between the computer and devices with the ASSIGN statement. Here is a typical example:

```
ASSIGN Any_name TO Device_selector
```

This assignment fills a "table" in memory with information that describes the I/O path. This information includes the device selector, the path's FORMAT attribute, and other descriptive information. When the I/O path name is specified in a subsequent I/O statement (such as OUTPUT or ENTER), this information is used by the system in completing the I/O operation.

Different default FORMAT attributes are given to devices and files:

- **Devices**—since most devices use an ASCII data representation, the default attribute assigned to devices is FORMAT ON. (This is also the default for ASCII files.)
- **BDAT** and **HP-UX** or **DOS** files—the default for BDAT and HPUX or DOS files is FORMAT OFF. (This is because the FORMAT OFF representation requires no translation time for numeric data; this is possible because humans never see the data patterns written to the file, and therefore the items do not have to be in ASCII, or humanly- readable, form.)

One of the most powerful features of this BASIC system is that you can change the attributes of I/O paths programmatically.

ManualsPlus.com

Specifying I/O Path Attributes

There are two ways of specifying attributes for an I/O path:

Specify the desired attribute(s) when the I/O path name is initially assigned. For example:

```
100  ASSIGN @Device TO Dev_selector; FORMAT ON
```

or

```
100  ASSIGN @Device TO Dev_selector ! Default for devices is FORMAT ON.
```

Specify only the attribute(s) in a subsequent ASSIGN statement:

```
250  ASSIGN @Device; FORMAT OFF  ! Change only the attribute.
```

The result of executing this last statement is to modify the entry in the I/O path name table that describes which FORMAT attribute is currently assigned to this I/O path. The implicit `ASSIGN @Device TO *`, which is automatically performed when the "`TO ...` " portion is included, is *not* performed. Also, the I/O path name must currently be assigned (in this context), or an error is reported.

Changing the EOL Sequence Attribute

In addition to the FORMAT attributes, another attribute is available to direct HP Instrument BASIC system to redefine the end-of-line sequence normally sent after the last data item in output operations.

An end-of-line (EOL) sequence is normally sent following the last item sent with free-field OUTPUT statements and when the "L" specifier is used in an OUTPUT that uses an image. The default EOL characters are carriage-return and line-feed (CR/LF), sent with no device-dependent END indication. You can also define your own special EOL sequences that include sending special characters, sending an END indication, and delaying a specified amount of time after sending the last EOL character.

In order to define non-default EOL sequences to be sent by the OUTPUT statement, an I/O path must be used. The EOL sequence is specified in one of the ASSIGN statements that describe the I/O path. Here is an example that changes the EOL sequence to a single line-feed character.

```
ASSIGN @File TO "file_one";EOL CHR$(10)
```

The characters following the secondary keyword `EOL` are the EOL characters. Any character in the range CHR$(0) through CHR$(255) may be included in the string expression that defines the EOL characters; however, the length of the sequence is limited to eight characters or less.

If END is included in the EOL attribute, an interface-dependent "END" indication is sent with (or after) the last character of the EOL sequence. However, if no EOL sequence is sent, the END indication is also suppressed. The following statement shows an example of defining the EOL sequence to include an END indication.

```
ASSIGN @Device TO 20;EOL CHR$(13)&CHR$(10) END
```

With the HP-IB Interface, the END indication is an End-or-Identify message (EOI) sent with the last EOL character.

The default EOL sequence is a CR and LF sent with no end indication. This default can be restored by using the EOL OFF attribute.

ManualsPlus.com

Restoring the Default Attributes

If any attribute is specified, the corresponding entry in the I/O path name table is changed (as above); no other attributes are affected. However, if no attribute is assigned (as below), then *all* attributes are restored to their default state (such as FORMAT ON for devices.)

```
340   ASSIGN @Device  !  Restores ALL default attributes.
```

Concepts of Unified I/O

The HP Instrument BASIC language provides the ability to communicate with the several system resources with the OUTPUT and ENTER statements.

The next section of this chapter describes how data can be moved to and from string variables with OUTPUT and ENTER statements. And, if you have read about mass storage operations in the "Data Storage and Retrieval" chapter of *HP Instrument BASIC Programming Techniques*, you know that the ENTER and OUTPUT statements are also used to move data between the computer and mass storage files.

This ability to move data between the computer and all of its resources with the same statements is a very powerful capability of the HP Instrument BASIC language.

Before briefly discussing I/O paths to mass storage files, the following discussion will present some background information that will help you understand the rationale behind implementing the two data representations used by the computer. The remainder of this chapter then presents several uses of this language structure.

Data-Representation Design Criteria

As you know, the computer supports two general data representations—the ASCII and the internal representations. This discussion presents the rationale of their design.

The data representations used by the computer were chosen according to the following criteria:

- to maximize the rate at which computations can be made
- to maximize the rate at which the computer can move the data between its resources
- to minimize the amount of storage space required to store a given amount of data
- to be compatible with the data representation used by the resources with which the computer is to communicate

The *internal representations* implemented in the computer are designed according to the *first three of the above criteria*. However, the last criterion must always be met if communication is to be achieved. If the resource uses the ASCII representation, this compatibility requirement takes precedence over the other design criteria. The *ASCII representation* fulfills this *last criterion* for most devices and for the computer operator. The first three criteria are further discussed in the following description of data representations used for mass storage files.

ManualsPlus.com

I/O Paths to Files

There are three types of *data files*: ASCII, BDAT, and HP-UX or DOS. Only the ASCII data representation is used with ASCII files, but either the ASCII (FORMAT ON) or the internal (FORMAT OFF) representation can be used with BDAT and HP-UX or DOS files.

BDAT, HPUX and DOS Files

BDAT, HP-UX and DOS files have been designed to maximize the efficiency with which HP Instrument BASIC moves, stores and manipulates data. Both numeric and string computations are much faster. These internal data representations allow much more data to be stored on a disc because there is no storage overhead (for numeric items), that is, there are no "record headers" for numeric items.

The **transfer rates** for each data type has also been *increased*. Numeric output operations are always much faster because there is no time required for "formatting". Numeric enter operations are also faster because the system does not have to search for item- and statement-termination conditions.

In addition, I/O paths to BDAT and HP-UX files can use either the ASCII (FORMAT ON) or the internal (FORMAT OFF) representation.

The following program shows a few of the features of BDAT files. The program first outputs an internal-form string (with FORMAT ON), and then enters the length header and string characters with FORMAT OFF.

```
110   DIM Length$[4],Data$[256],Int_form$[256]
120   !
130   ! Create a BDAT file (1 record; 256 bytes/record.)
140   ON ERROR GOTO Already_created
150   CREATE BDAT "B_file",1
160   Already_created: OFF ERROR
170   !
180   ! Use FORMAT ON during output.
190   ASSIGN @Io_path TO "B_file";FORMAT ON
200   !
210   Length$=CHR$(0)&CHR$(0)  ! Create length header.
220   Length$=Length$&CHR$(0)&CHR$(252)
230   !
240   ! Generate 256-character string.
250   Data$="01234567"
260   FOR Doubling=1 TO 5
270      Data$=Data$&Data$
280   NEXT Doubling
290   ! Use only 1st 252 characters.
300   Data$=Data$[1,252]
310   !
320   ! Generate internal-form and output.
330   Int_form$=Length$&Data$
340   OUTPUT @Io_path;Int_form$;
350   ASSIGN @Io_path TO *
360   !
370   ! Use FORMAT OFF during enter (default).
380   ASSIGN @Io_path TO "B_file"
390   !
```

(Continued)

ManualsPlus.com

```
400   ! Enter and print data and # of characters.
410   ENTER Data$
420   PRINT LEN(Data$);"characters entered."
430   PRINT
440   PRINT Data$
450   ASSIGN @Io_path TO * ! Close I/O path.
460   !
470   END
```

ASCII Files

ASCII files are designed for interchangeability with other HP computer systems. This interchangeability imposes the restriction that the data must be represented with ASCII characters. Each data item sent to these files is a special case of FORMAT ON representation; *each item is preceded by a two-byte length header* (analogous to the internal form of string data). In order to maintain this compatibility, there are two additional restrictions placed on ASCII files:

- The FORMAT OFF attribute *cannot* be assigned to an ASCII file
- You cannot use OUTPUT..USING or ENTER..USING with an ASCII file.

The following program shows the I/O path name @Io_path being assigned to the ASCII file named ASC_FILE. Notice that the file name is in all uppercase letters; this is also a compatibility requirement when using this file with some other systems.

The program creates an ASCII file, and then outputs program lines to the file. The program then gets and runs this newly created program. (If you type in and run this program, be sure to save it on disc, because running the program will load the program it creates, destroying itself in the process.)

```
100   DIM Line$(1:3)[100]  ! Array to store program.
110   !
120   ! Create if not already on disc.
130   ON ERROR GOTO Already_exists
140   CREATE ASCII "ASC_FILE",1  ! 1 record.
150   Already_exists:  OFF ERROR
160   !
170   ASSIGN @Io_path TO "ASC_FILE"
180   STATUS @Io_path,6;Pointer
190   PRINT "Initially:    file pointer=";Pointer
200   PRINT
210   !
220   Line$(1)="100  PRINT ""New program.""  "
230   Line$(2)="110  BEEP"
240   Line$(3)="120  END"
250   !
260   OUTPUT @Io_path;Line$(*)
270   STATUS @Io_path,6;Pointer
280   PRINT "After OUTPUT: file pointer=";Pointer
290   PRINT
300   !
310   GET "ASC_FILE" ! Implicitly closes I/O path.
320   !
330   END
```

ManualsPlus.com

Data Representation Summary

The following table summarizes the control that programs have on the FORMAT attribute assigned to I/O paths.

Program Control of the FORMAT Attribute

Type of Resource	Default FORMAT Attribute Used	Can Default FORMAT Attribute Be Changed?
Devices	FORMAT ON	Yes (if an I/O path is used)[1]
BDAT files	FORMAT OFF	Yes
HP-UX or DOS files	FORMAT OFF	Yes
ASCII files	FORMAT ON[2]	No
String variables	FORMAT ON	No

[1]FORMAT ON is *always* used whenever an OUTPUT ... USING or ENTER ... USING statement is used, regardless of the FORMAT attribute assigned to the I/O path.
[2]The data representation used with ASCII files is a special case of the FORMAT ON representation.

Applications of Unified I/O

This section describes two uses of the powerful unified-I/O scheme of the computer. The first application contains further details and uses of I/O operations with string variables. The second application involves using a disc file to simulate a device.

I/O Operations with String Variables

This section describes both the details of and several uses of outputting data to and entering data from string variables.

Outputting Data to String Variables

When a string variable is specified as the destination of data in an OUTPUT statement, source items are evaluated individually and placed into the variable according to the free-field rules or the specified image, depending on which type of OUTPUT statement is used. Thus, item terminators may or may not be placed into the variable. The ASCII data representation is always used during outputs to string variables.

Characters are always placed into the variable beginning at the first position; no other position can be specified as the beginning position at which data will be placed. Thus, *random access of the information in string variables is not allowed* from OUTPUT and ENTER statements; all data must be accessed serially. For instance, if the characters "1234" are output to a string variable by one OUTPUT statement, and a subsequent OUTPUT statement outputs the characters "5678" to the same variable, the second output *does not* begin where the first one left off (i.e., at string position five). The second OUTPUT

ManualsPlus.com

statement begins placing characters in position one, just as the first OUTPUT statement did, overwriting the data initially output to the variable by the first OUTPUT statement.

The string variable's length header (4 bytes) is updated and compared to the dimensioned length of the string as characters are output to the variable. If the string is filled before all items have been output, an error is reported; however, the string contains the first n characters output (where n is the dimensioned length of the string).

Example

The following program outputs string and numeric data items to a string variable and then calls a subprogram that displays each character, its decimal code, and its position within the variable.

```
100   ASSIGN @Crt TO 1  ! CRT is disp. device.
110   !
120   OUTPUT Str_var$;12,"AB",34
130   !
140   CALL Read_string(@Crt,Str_var$)
150   !
160   END
170   !
180   !
190 SUB Read_string(@Disp,Str_var$)
200     !
210     ! Table heading.
220     OUTPUT @Disp;"----------------------"
230     OUTPUT @Disp;"Character  Code  Pos."
240     OUTPUT @Disp;"---------  ----  ----"
250     Dsp_img$="2X,4A,5X,3D,2X,3D"
260     !
270     ! Now read the string's contents.
280   FOR Str_pos=1 TO LEN(Str_var$)
290         Code=NUM(Str_var$[Str_pos;1])
300       IF Code<32 THEN ! Don't disp. CTRL chars.
310           Char$="CTRL"
320       ELSE
330           Char$=Str_var$[Str_pos;1] ! Disp. char.
340       END IF
350       !
360       OUTPUT @Disp USING Dsp_img$;Char$,Code,Str_pos
370   NEXT Str_pos
380   !
390   ! Finish table.
400   OUTPUT @Disp;"----------------------"
410   OUTPUT @Disp ! Blank line.
420   !
430   SUBEND
```

ManualsPlus.com

```
---------------------
Character  Code  Pos.
---------  ----  ----
             32     1
1            49     2
2            50     3
,            44     4
A            65     5
B            66     6
CTRL         13     7
CTRL         10     8
             32     9
3            51    10
4            52    11
CTRL         13    12
CTRL         10    13
---------------------
```

Outputting data to a string and then examining the string's contents is usually a more convenient method of examining output data streams than using a mass storage file. A string may contain both printing and non-printing (control) characters. Printing string contents that contain control characters could interfere with examining the data stream. The preceding subprogram may facilitate viewing this data without viewing such strings.

Example

Outputs to string variables can also be used to generate the string representation of a number, rather than using the VAL$ function (or a user-defined function subprogram). The *main advantage* is that you can explicitly specify the number's image while still using only a single program line. The following program compares the string generated by the VAL$ function to that generated by outputting the number to a string variable.

```
100   X=12345678
110   !
120   PRINT VAL$(X)
130   !
140   OUTPUT Val$ USING "#,3D.E";X
150   PRINT Val$
160   !
170   END

1.2345678E+7   Printed results
123.E+05
```

Entering Data From String Variables

Data are entered from string variables in much the same manner as output to the variable. All ENTER statements that use string variables as the data source interpret the data according to the FORMAT ON attribute. Data is read from the variable beginning at the first string position; if subsequent ENTER statements read characters from the variable, the read also begins at the first position. If more data are to be entered from the string than are contained in the string, an error is reported; however, all data entered into the destination variable(s) before the end of the string was encountered remain in the variable(s) after the error occurs.

ManualsPlus.com

When entering data from a string variable, the computer keeps track of the number of characters taken from the variable and compares it to the string length. Thus, *statement-termination* conditions are *not* required; the ENTER statement automatically terminates when the last character is read from the variable. However, *item* terminators are still required *if* the items are to be separated *and* the lengths of the items are not known. If the length of each item is known, an image can be used to separate the items.

Example

The following program shows an example of the need for *either* item terminators *or* length of each item. The first item was not properly terminated and caused the second item to not be recognized.

```
100   OUTPUT String$;"ABC123";  ! OUTPUT w/o CR/LF.
110   !
120   ! Now enter the data.
130   ON ERROR GOTO Try_again
140   !
150 First_try: !
160   ENTER String$;Str$,Num
170   OUTPUT 1;"First try results:"
180   OUTPUT 1;"Str$= ";Str$,"Num=";Num
190   BEEP     ! Report getting this far.
200   STOP
210   !
220 Try_again: OUTPUT 1;"Error";ERRN;" on 1st try"
230            OUTPUT 1;"STR$=";Str$,"Num=";Num
240            OUTPUT 1
250            OFF ERROR  ! The next one will work.
260            !
270   ENTER String$ USING "3A,3D";Str$,Num
280   OUTPUT 1;"Second try results:"
290   OUTPUT 1;"Str$= ";Str$,"Num=";Num
300   !
310   END
```

This technique is convenient when attempting to enter an unknown amount of data or when numeric and string items within incoming data are not terminated. The data can be entered into a string variable and then searched by using images.

Example

ENTERs from string variables can also be used to generate a number from ASCII numeric characters (a recognizable collection of decimal digits, decimal point, and exponent information), rather than using the VAL function. As with outputs to string variables, images can be used to interpret the data being entered.

```
30    Number$="Value= 43.5879E-13"
40    !
50    ENTER Number$;Value
60    PRINT "VALUE=";Value
70    END
```

ManualsPlus.com

Index

ManualsPlus.com

ManualsPlus.com

ManualsPlus.com