Personal Software Division

HP 150 Software Development Tools For Independent Software Vendors

HEWLETT PACKARD

3410 CENTRAL EXPRESSWAY, SANTA CLARA, CA 95051

Printed in U.S.A. 2/84

NOTICE

PRINTING HISTORY

New editions are complete revisions of the manual. Update packages, which are issued between editions, contain additional and replacement pages to be merged into the manual by the customer. The dates on the title page change only when a new edition or a new update is published. No information is incorporated into a reprinting unless it appears as a prior update; the edition does not change when an update is incorporated.

The software code printed alongside the date indicates the version level of the software product at the time the manual or update was issued. Many product updates and fixes do not require manual changes and, conversely, manual corrections may be done without accompanying product changes. Therefore, do not expect a one to one correspondence between product updates and manual updates.

First Edition . . . . Feb 8, 1984. . . . . A.01.00

Table of Contents

INTRODUCTION

ABOUT THIS BOOKLET

This booklet tells you about the tools available for writing software on the HP 150. Some of these tools are an integral part of the standard HP 150 system, and some are programs/utilities contained on the ISV Development Disc. We hope that these tools help you to convert existing software or develop new software with the least amount of effort. The next section, "Procedures for Getting Started" gives you some guidelines on how to proceed when you are converting your software to the HP 150. You should review this section before proceeding on to other sections.

The material in this booklet is presented in a manner that assumes that you are familiar with assembly language programming for the Intel 8086/8088 chip. If you are not, we suggest that you purchase any of several excellent books available at your local bookstore or computer retailer. The *MS-DOS Programmer's Reference Manual* is included in your ISV Development Pack. It augments the HP 150 specific material presented in this booklet. You may need to refer to the HP 150 manuals available to supplement the material in this booklet. For example, The *HP 150 Personal Computer Owner's Guide* gives details about P.A.M. which is useful when you're running your application in the standard environment.

About The HP 150

You should understand one aspect of the HP 150 before you begin programming your application. This aspect is the dual nature of the HP 150 itself. Unlike many personal computers, the HP 150 functions as a stand-alone personal computer as well as a full-function intelligent terminal.

Because the functionality is greater, you have more flexibility in writing your application programs. At the same time, to use this increased capability to the fullest extent, you may need to spend extra time familiarizing yourself with the various capabilities available.

Since the HP 150 is both a personal computer as well as a full function terminal, there are two ways your program can control the display and keyboard. Hewlett-Packard supports:

- Escape sequence programming
- AGIOS Function Calls

The first, using escape sequence sequences, is the easiest to implement and can be used in standard console output statements in any programming language. In this method, sending an escape character (ASCII code 27 decimal) causes the HP 150 terminal to interpret the next one or more characters as special control codes, not as characters to be displayed.

There is a price to pay for such ease: the escape sequence interpreter is relatively slow. Standard console output occurs at a rate of approximately 700 characters a second. If this speed is acceptable, you will probably want to use escape sequence programming.

The second method, called AGIOS, is unique to the HP 150. To use AGIOS, you normally issue an assembly language cal, not standard input/output statements. You might wonder why you would want to use AGIOS if it requires a special set up. The answer is system-dependent speed: AGIOS allows console output run as fast as 4000 characters a second!

This AGIOS, which stands for "Alpha Graphics Input/Output Subsystem", is provided so that you can write high-speed output routines and gain total control over the HP 150.

You will not find AGIOS on other micro-computers. Most micros require you to use hardware specific information on that system in order to make your application faster. This makes your program dependent on specific hardware addresses, revisions of computer firmware (ROM) and the operating system BIOS. If the manufacturer changes some small detail in the computer, your application may not run and your customers will be calling you.

Hewlett-Packard is committed to maintaining compatibility in future systems in the Series 150 line. This includes support for both AGIOS and escape sequence programming. When your software is optimized using these facilities, it should run with few, if any, modifications on future products like the HP 150.

Because of this commitment to future compatibility, HP has elected not to document hardware details, such as port addresses and BIOS specifics. While this may seem strange, you should find that AGIOS, escape sequences, and standard MS-DOS functions make your software compatible in the future, reducing your investment in program maintenance.

PROCEDURES FOR GETTING STARTED

SECTION 2

Introduction

The purpose of this section is to give you guidelines on how to get your application up as quickly as possible. As the guidelines are presented, you are told where you can find related reference material, either in this booklet, or in other documents.

1. Determine which programming language you're going to use. Appendix D contains a summary of the languages which should work on the HP 150.

   You'll need to purchase a copy of the language you're going to use. If it is an HP-supported language, you can purchase it on HP media from your local retailer. If the language is produced by another manufacturer, you may need to buy it in IBM disc format and copy it to the HP 150. Refer to the section, "Transferring Software to the HP 150" for instructions on how to transfer your language files to the HP 150.

2. Study the material in this booklet about escape sequences and AGIOS function calls. Decide which method (or combination of the two) you're going to use in your application programs.

   If you're converting existing programs to the HP 150, you should take the time to investigate substituting AGIOS function calls for some or all of your screen and keyboard I/O. While there is more set-up time required initially, the execution time is extremely fast (for example, block text operations reach 4000 char./sec).

   Escape Sequences and AGIOS function calls are explained in Sections 1, 4 and 5.

3. If you're converting existing software to the HP 150, follow the procedures in section 3 to transfer your program files to the HP 150.

4. If you're converting your software, you have to modify all hardware-specific calls. Some of the things to watch for are:

   Graphics Screen Operations

   Refer to Sections 4 and 5 in this booklet for details on graphics operations, such as vector plotting and graphics character set operations.

   Interrupts

   Use only the standard MS-DOS functions that are described in the *MS-DOS Programmer's Reference Manual*. For example, INT 10 is a standard keyboard interrupt on the IBM PC, but is not a standard MS-DOS interrupt.

   Calls to BIOS

   The HP 150 BIOS source listing is not available. Virtually all of the tasks which commonly require BIOS calls on other micros can be done using AGIOS. See Sections 5 and 7 for details.

I/O Port Control

Because of Hewlett-Packard's commitment to compatibility, we generally do not discuss specific port maps. MS-DOS lets you read from and write to devices as files. Refer to Section 7 for details.

Terminal Control Codes

The HP 150 uses the standard HP 2623 escape sequences which are not entirely identical to the ANSI standards. You need to check the escape sequences used in your program, if any, and change those that are different from the HP 150. Appendix A gives a quick reference to the HP 150 escape sequences.

Keyboard Control

The HP 150 offers unique control of keyboard operations. Keyboard input is not interrupt driven within MS-DOS. See Section 7 for details.

IF YOU NEED TECHNICAL ASSISTANCE

Remember that, as a qualified Independent Software Vendor, you can get technical assistance from Hewlett-Packard regarding subjects discussed in this booklet. Specifically, those subjects are:

- AIOS/GIOS function calls
- Escape sequence programming
- General technical questions about the HP 150

Assistance *cannot* be provided for these general areas:

- Applications Consulting
- Language Consulting
- Source code listings
- Specific subroutines or subprograms
- Hardware Consulting
- Specific consulting on your application
- Information on other vendor's applications
- Advice concerning applications in general
- Support for non-HP peripherals

The assistance line for ISVs is staffed during normal business hours (West Coast Time). There is an answering machine on the line in case no one is available when you call. Our phone number is listed on the cover letter of this booklet. When you call, please have your ISV Software Development Agreement Number handy. It is required for technical assistance.

You can also get technical assistance via electronic mail, and we suggest that you do this when possible. By using electronic mail, you often get much more descriptive information and get it more conveniently than by phone. To use electronic mail, you need to be a subscriber to *The Source*. If you do not currently have this service, you can sign up at your local computer retailer. The HP account number on *The Source* is TCT878.

THIS PAGE SHOULD BE BLANK

TRANSFERRING YOUR SOFTWARE TO THE HP 150

SECTION 3

INTRODUCTION

Because the HP 150 does not support "industry standard" disc formats, you need to transfer your software to the HP 150 via data communications facilities.

The ISV Development Disc contains several utilities to transfer your files. The one that you use depends on the computer you want to transfer from, as well as the kind of file you're transferring.

Transferring From:	Source File Types:	Utilities to Use:
IBM PC	ASCII, Binary	DSNLINK/150 and MONITOR/PC
IBM PC	ASCII, Binary	KERMIT/150 and KERMIT/PC
Other PC's	ASCII	DSNLINK/150 (logging feature)

NOTE

If you need to transfer binary files, see "Communicating Between Two Workstations With DSN/MONITOR" in the *Series 100/DSN/LINK* manual. This manual suggests ways you can write your own MONITOR utility.

Other PC'S	ASCII, Binary	XMODEM/150

NOTE

You need a comparable XMODEM program on your particular PC. XMODEM uses the "Ward Christensen" protocol, which is available in the public domain for a variety of PC's. The source code for XMODEM is also included on the ISV Development Disc. If you cannot find a comparable program for your PC, you can use XMODEM as a guide and develop your own program.

For detailed instructions on using the above utilities, refer to the appropriate paragraphs under "Starting the Transfer".

Cabling Your PC to the HP 150

To transfer your software to the HP 150, you need a cable link from your PC to the HP 150. The cable you need is a simple male to male RS-232 cable with pins 2 and 3 switched and with other signals required for hardware handshaking. The significant signals are pins 2, 3, and 7. These pins are for the IBM PC. If you have another computer, you may need to change these assignments.

If you are building your own cable, use the pin assignments:

```
        IBM PC End              HP 150 End
        ----------              ----------

            2   - - - - - - - - - - -   3
            3   - - - - - - - - - - -   2
            7   - - - - - - - - - - -   7
```

You will also need to jumper pins 5, 6, and 20 on the HP 150 end depending on the handshaking options you have selected on your system. No handshaking is required for use of the utilities described in this section.

If you would rather not build your own, Hewlett-Packard can provide the appropriate cable. Our supplies service can be reached at 800-538-8787 (in California 408-738-4133 collect). You can use a single cable (HP Part Number 13242H) or two in combination (HP Part Numbers 13242N, 13232U). The 13242H cable does the job, but is often not available in the same quantities as the 13242N and 13232U cables.

Configuring the HP 150

When you're transferring files to the HP 150, the baud rate on the sending PC and the HP 150 must be the same. The default baud rate on the HP 150 is 2400. If you want to change this rate, follow these procedures:

1. If you're in P.A.M., exit to MS-DOS.
2. Touch User/System twice.
3. Touch Config Keys [f8].
4. Touch Global Config [f1].

Determine which port is configured as the "Remote" port (as opposed to the "Serial" port). Next, while still in the Global Config display:

5. Touch Config Keys [f8].
6. Touch either Port1 [f3] or Port2 [f4] depending on which port is your data comm port.
7. Touch NEXT CHOICE [f2] until you see the baud rate you need (9600 is probably the optimum).
8. Touch SAVE CONFIG [f1].

You also need to use the Device Configuration Utility to configure COM1 to the selected "Remote" device. Refer to the *HP 150 Personal Computer Owner's Guide* for instructions on how to do this.

Starting the Transfer

The following paragraphs tell you how to perform the transfer between your computer and the HP 150. Check at the beginning of this section to determine which utilities you need to use on the HP 150 to perform the transfer.

USING DSN/LINK WITH THE IBM PC.

1. Be sure the HP 150 is configured for the baud rate you're going to use. You'll probably want to use 9600 baud if you're connecting the two systems directly. Telephone connections require lower speeds.

On Your IBM PC:

2. Boot your IBM PC as usual using your standard operating system.

   You need to use version 2.0 or higher of PC-DOS. You also need one double-density disc drive.

3. Run the MONITOR program provided on the ISV Development Disc.

4. Alter the baud rate in MONITOR if you're using a rate other than 9600. To do this, press Config [f2]. When you have selected the appropriate rate, type [f8] to exit the IBM configuration menu.

5. Insert the application disc that you want to transfer into one of your IBM disc drives. MONITOR runs unattended from this point.

<u>On the HP 150:</u>

6. Run DSNLINK/150. Touch Transfer From Host [f2]. Complete the form, specifying the file name on the IBM PC and the file name on the HP 150. Be sure to include the disc drive letter if different from the default disc.

7. If the file is binary (for example, a compiler) press the ASCII/Binary softkey [f3], then press Start Transfer [f1] when you're ready to begin.

8. Repeat steps 6 and 7 for each file you want to transfer.

9. When all files are transferred, exit DSNLINK/150. DSNLINK/150 terminates the MONITOR program on your IBM PC.

USING KERMIT WITH THE IBM PC.

1. Be sure the HP 150 is configured for the baud rate you're going to use. You'll probably want to use 9600 baud if you're connecting the two systems directly. Telephone connections require lower speeds.

On Your IBM PC:

2. Boot your IBM PC as usual using your standard operating system.

   You need to use version 2.0 or higher of PC-DOS. You also need one double-density disc drive.

3. Run KERMIT/PC from the ISV Development Disc.

4. If you're not going to use a 1200 baud rate, type the baud rate you want to use. You do this by typing, "Set Baud xxxx", and substituting the number you want to use in the place of "xxxx".

On the HP 150

5. Use the Configuration Menu to change the speed to the rate you're going to use. You cannot use KERMIT/150 to change the baud rate.

6. From MS-DOS, run KERMIT/150. At the KERMIT prompt, type "Receive". To direct files to other than the default drive, type the drive letter you want to use. The sending program passes the file name. For example, to receive files on disc drive C, type "Receive C:".

7. From the IBM PC, type the disc drive location and file name of the file you want to transmit. For example, if you want to transfer all files on drive A, type "Send A:*.*".

   When KERMIT PC is finished, you see "Completed" on the IBM display. To transfer additional files, repeat steps 6 amd 7.

TRANSFERRING FROM OTHER PC'S.

You can perform ASCII file transfers from PC's other than the IBM PC by using DSNLINK/150. You are going to use the "logging" option of this program.

1. Be sure the HP 150 is configured for the baud rate you're going to use. You'll probably want to use 9600 baud if you're connecting the two systems directly. Telephone connections require lower speeds.

On the HP 150

2. Run DSNLINK/150 and select Logging Options [f4]. Now select Local File [f1] and type the local file name. All activity on the data comm line is now sent to that file, so whatever is sent by the other PC will be stored.

3. Select Local File to stop. If you want to log another file, select Local File again, and type the new file name.

THIS PAGE SHOULD BE BLANK

ESCAPE SEQUENCE PROGRAMMING

SECTION 4

INTRODUCTION

Escape sequence programming is probably the easiest method of programmatically controlling the HP 150 keyboard and display.

This section gives details and examples about escape sequence programming. This information is intended to help you with the most unique features of the HP 150. The topics included in this section are:

- User Definable Softkeys
- Touch Screen Programming
- Keyboard Control
- Character Enhancements and Screen Control
- Escape Sequence Syntax Summary

The last part of the section contains excerpts from the *HP 150 Advanced User's Guide*. At the time of this writing, the *HP 150 Advanced User's Guide* is in the preliminary editing phase. This means the material has not been through final review. If you have questions about the material, you need to refer to the completed manual when it becomes available. The material suggests a number of features that can be used from a remote host. In fact, any MS-DOS application can use these features as well. Notes in this section explain the material presented as it applies to a local programming environment.

You will also find a brief summary of escape sequences in the *HP 150 Terminal Users Guide* which was included with your system. Appendix A of this document also contains such a summary.

INTRODUCTION TO SOFTKEYS

There are two distinct sets of softkeys on the HP 150. While you can use both sets in a single application, you will probably want to select one or the other. The first set is the "User-Definable Softkeys" programmed via escape sequences. The other set, "Application Softkeys", are controlled via AGIOS calls. The following summary gives you more details on these two sets of softkeys.

User-Definable Softkeys

- The set of 8 keys labeled [f1] through [f8]
- Are defined and displayed under program control (escape sequences).
- Are accessed by typing [Shift] [User System].
- Can contain up to 80 characters.
- Can be programmed to perform terminal functions only, or to simulate keyboard input with or without an automatic carriage return.
- Display the default label (i.e., [f1]) or any label up to 16 characters on two lines.

Application Softkeys

- The set of 8 keys labeled [f1] through [f8] and the the 4 unlabeled keys at the top of the numeric keypad.
- Are defined, displayed, and controlled by AGIOS function calls.
- Are normally used for input only in "keycode mode". See "Keycodes" in Section 7 for more information.
- Return a keycode for each key, not the buffer associated with the "User-Definable Softkeys".
- Are displayed by the user by typing [CTRL] [User System].

For more information on Application Softkeys, refer to "Application Softkeys" in Section 5.

User-Definable Softkeys

The escape sequence that you use to program the User-Definable set of softkeys has the following general format:

```
ESC & f <attr> a <key> k <lab len> d <buf len> L <lab> <buf>
```

As is true with all escape sequences, the HP 150 processes this sequence until the first upper case character is encountered. The characters in the above escape sequence are defined as:

`ESC`	The escape character, ASCII 27 decimal.
`& f`	The "define function key" sequence.
`<attr> a`	The attribute. Values for `<attr>` are: 0 = Normal Key 1 = Local Key 2 = Transmit Key See the note which follows.
`<key> k`	The softkey number. Key must be between 1 and 8.
`<lab len> d`	The length of the softkey label `<lab>`.
`<buf len> L`	The length of the response buffer `<buf>`: must be between -1 and 80 Notice that the character that follows `<buf len>` is an uppercase "l". This ends the escape sequence and the `<lab>` and `<buf>` parameters which follow it are treated as data when the escape sequence is executed. If you do not want to change the `<buf len>` parameter, make sure that the "d" which ends `<lab len>` is uppercase. A length of -1 clears the existing buffer.
`<lab>`	The ASCII characters to be displayed in the on-screen softkey labels.
`<buf>`	The ASCII characters associated with the softkey.

NOTE

Local defined keys execute only in the HP 150 display. None of the characters in `<buf>` will be sent to the user program. Generally, local keys will display on the screen but not get to console input. Local keys are generally not useful within applications.

Normal and Transmit keys are both sent to standard console input. The principal difference is that in Transmit keys a carriage return is appended to the end of each softkey (hence transmitted to the internal computer). The characters in a Normal key buffer are sent to the program, but no carriage return is automatically appended.

In BASIC, for example, the INPUT statement requires a carriage return to end input. If you are using the INPUT statement, be sure to define keys with the Transmit attribute.

However, the INPUT$ function does not require a carriage return. If you are using this instruction for input, you may elect to use Normal keys.

The sum of `<lab len>` and `<buf len>` must be less than 160 characters.

Displaying the User-Definable Softkeys

Once you define one (or more) of the User-Defined Softkeys, you will probably want to display them. To do so, use the following general escape sequence format, with the appropriate `<parm>` value:

```
ESC & j <parm>
```

The `<parm>` options are:

```
@ = Turn off screen labels, time display, and status line.
A = Display the "System" softkeys.
B = Display the User-Defined softkeys.
S = Lock the currently displayed set of softkeys on the screen.
R = Unlock the softkeys so that they can be redefined or changed.
```

The meanings of each of the <parm> field follows:

@:	This option turns off the current screen displayed labels and the row and column indicators. It also turns off the Status Line, including the time and other status information.
A:	This options causes the display of the User Defined Softkey labels on the screen. This is the level a user would see by pressing [User/System].
B:	With this option, the system displays the currently defined set of User Defined Softkey labels. If no labels have been defined, the default labels [f1] through [f8] are used. This is the programmatic equivalent of [CTRL] [User/System].
S:	This selection causes the currently displayed set of screen labeled keys to be 'locked' in place. Any attempt to change them (by a user or programmatically) will cause the system to 'beep' and the labels to remain intact.

R: This option unlocks the softkey labels so that they may be changed (by a user or programmatically).

There is an irregularity in the HP 150 firmware such that User- Definable Softkeys are "live" to touch, even when the labels are not displayed. That is, when the User Defined SoftKeys are displayed using the `ESC & j B` sequence, and then 'turned off' with the `ESC & j @` sequence, the screen labels are still active and will return the softkey definition when touched.

Be sure to account for this possibility in your application when you design your program. Do this by always expecting touch input in your application, or by specifically disabling touch on the softkey area. This latter alternative is presented in `Controlling User-Defined Softkeys` in the next few pages.

You will find that, when you use the ESC & f sequence, the changed labels/buffers do not "take effect" until you re-display the labels with an ESC & j B sequence.

If you are using "Keycode Mode" for input and User Defined Softkeys, remember that you receive four bytes of input for EACH character in the `<buf>` field! See the section on "Keycode Mode" for more information.

There is one more valid parameter to the `ESC & j` sequence which requires slightly different syntax. The syntax of this sequence is:

```
ESC & j <len> L <message>
```

This sequence allows an application to display up to 160 characters (two lines) of text in place of the softkey labels. The message is displayed on the screen until the [Return] key is pressed, and cannot be locked on the screen. The `<len>` parameter must be between 0 and 160 inclusive, and specifies the number of bytes after the `L` are to be displayed. The `<message>` buffer will be displayed in place of the screen labeled softkeys.

Programming Considerations

User-Defined Softkeys on the HP 150 are not interrupt driven as they are on many micros. This means that you need to 'poll' console input for softkey presses.

There are times you will want softkey input to be indistinguishable from keyboard input. For example, an application menu may prompt the user for a numbered selection. By defining the softkey contents so that the selection number corresponds to the key number, the user can either type the menu selection or press the appropriate softkey. This allows the User Defined Softkeys, labeled appropriately, to be used to augment other keyboard input.

At other times, you may want softkey input to be unique. For example, an application may offer a user the ability to either type a filename or to exit the program. If the user types any valid string, the application considers that input a filename. If the 'exit' softkey is pressed, the application ends.

One way to solve this dilemma is to define the softkeys with control characters. While the user may enter a control character from the console, you can nonetheless distinguish between normal ASCII input and the control-code definition of the softkeys.

To use such a scheme successfully, you need to select softkey definitions which do not have unwanted affects on the operation of the system. ASCII values in the range of 18 through 25 can be used without any such interference. In lower ranges, tab (8), line feed (10), carriage return (12), and DC1 (17) interfere. Above this range, the escape character (27) interferes.
Given softkey definitions in that range, here is a BASIC program segment which can accept keyboard input and set a flag (KEY) to the softkey number when a softkey is pressed.

```
3000 A$ = INPUT$(1)             'Get character without echo
3010 A = ASC(A$) : KEY = 0      'Get value; assume no KEY
3020 IF A<18 OR A>25 THEN 3050  'Not in softkey range
3030 KEY = A - 17               'Put KEY in range 1-8
3040 'Continue with program
```

Remember: the User Defined Softkeys can be used to your advantage
to make your applications easier to use.

USING TOUCH SCREEN

One of the most unique features of the HP 150 is the touch sensitive screen. The touch screen can be programmed in a variety of ways, giving you flexibility in designing your application. This section tells you about touch screen, how you program it, and finally how you can make the best use of it in your application.

The first thing you need to know about programming the touch screen is the "modes" of operation. The simplest and most flexible mode of operation involves "fields". In this mode, you tell the HP 150 information about where you want to detect touches, and the system firmware does most of the work.

As you will see, you can turn the touch screen on and off without affecting the fields you may have defined. You can also select video enhancements for both "touched" and "untouched" fields, and control cursor positioning and beeping.

A more comprehensive solution, but one which requires more work by the application, involves "row and column sensing". In this mode, the touch screen passes absolute row and column addresses to your application. As with fields, you can turn sensing on and off, and control cursor positioning and beeping.

Defining a Touch Field

The general form of a touch field definition is:

```
ESC - z g <rows> r <cols> c  <curs> c <beep> b      /
          <on_enh> e <off_enh> f  <attr> a  <mode> m  /
          <buf len> L <buf>
```

The meaning and valid entries for each of these fields follows. Note that the "/" character is the continuation line marker, not part of the sequence!

`ESC`	The escape character, ASCII 27 decimal.
`-zg`	The sequence which represents a touch field definition.
`<rows> r`	Specifies the beginning and ending rows for this touch field. The `<rows>` parameter is specified as: `<Start-row , End-row>` The valid ranges for `Start-row` and `End-row` are numeric ASCII strings from 0 to 23 inclusive. `End-row` must be greater than or equal to the `Start-row`.
`<cols> c`	Specifies the beginning and ending columns for this touch field. The `<cols>` parameter is specified as: `<Start-col , End-col>` The valid ranges for `Start-col` and `End-col` are numeric ASCII strings from 0 to 79 inclusive. `End-col` must be greater than or equal to the `Start-col`.
`<curs> p`	This parameter specifies whether the alpha cursor should be positioned at the upper left corner of the field. Possible values for `<curs>` are: 0 = The cursor does not move to the field. 1 = The cursor is positioned at the upper left corner of the field. If "p" is not specified, the cursor is not positioned on touch.
`<beep> b`	The "b" parameter specifies whether the system should beep when the field is touched. Valid values for `<beep>` are: 0 = No sound occurs when the field is touched. 1 = The system beeps on touch. If "b" is not specified, no beeping occurs when touched.
`<on_enh> e`	This parameter specifies the enhancement (if any) which is displayed when the field is off (not touched). Possible values for `<on_enh>` are shown in this table: If "e" is not specified, the default on-enhancement of 10 is used. This displays a half-bright inverse field.
`<off_enh> f`	This specifies the enhancement displayed when the field is on, or touched. The possible values for `<off enh>` are shown in the "Touch Enhancements" table above. If "f" is not specified, the default off-enhancement of 2 is used. This causes an inverse video enhancement in the field.

Touch Enhancements Table:

`<parm>` value :	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Half-bright :									X	X	X	X	X	X	X	X
Underline :					X	X	X	X					X	X	X	X
Inverse Video:			X	X			X	X			X	X			X	X
Blinking :		X		X		X		X		X		X		X		X

`<attr> a`

This parameter specifies the type of field to be defined. The possible values for `<attr>` are:

1 = ASCII field
2 = keycode field
3 = toggle field
4 = normal field

For a full explanation of the types of fields refer to the next section, "Types of Touch Fields".

`<mode> m`

This parameter specifies the sensing mode for this field. The valid values for `<mode>` are:

1 = Report on touch.
2 = Report on release.
3 = Report on touch and release.

As you would assume, this parameter determines *when* the HP 150 "tells" your application that touch is sensed. The section, "Specifying Reporting Modes" gives you more information about this parameter.

`<buf len> L`

This parameter specifies the length of the response string associated with this field.

`<buf>`

The response buffer to be associated with this field. `<buf>` may be 0 to 80 bytes for ASCII fields but must be two characters in length for Toggle and Normal fields.

Types of Touch Fields

The HP 150 lets you define several types of touch fields. The discussion which follows presents all of the types, although you will find some are more appropriate than others for use within BASIC. While this does limit your options, it does not prevent you from fully utilizing the touch features within your application.

The four types of touch fields are:

1. ASCII Fields: These are very similar to the User-Definable Softkeys in that a buffer of up to 80 characters can be associated with every field. In an ASCII field, each time touch is sensed (see "Specifying Reporting Modes") your application receives the designated buffer as standard console input.

2. Keycode Fields: These fields require Keycode Mode which is not easily done within high level languages. In this mode, a touch field simulates the typing of keys on the keyboard. Refer to "Keycode Modes" in Section 7 for information on using these fields.

3. Toggle Fields: Toggle fields are "regions" for which on and off states make sense. For example, you might permit the user to enter any of several options, and act on the selected fields only when [Return] is typed. The following escape sequence is used for toggle fields:

   ```
   ESC - z <buf> <type> Q
   ```

   <Buf> is the two-character buffer specified when the toggle field is defined. The <type> parameter will return:

   1 = Toggle field turning on.
   2 = Toggle field turning off.

4. Normal Fields: These fields are similar to ASCII fields, but you use only a two-character <buf> (same as toggle fields). The following escape sequence is used for normal fields:

   ```
   ESC - z  <buf>  <type> Q
   ```

   The <buf> parameter is the two character buffer specified when the field is defined. Valid return values for <type> are:

   5 = Normal field touch sensed
   6 = Normal field release sensed

Again, the main difference between Toggle Fields and Normal Fields is that Toggle Fields have two states, on and off. A Normal Field is "on" only while it is actually touched.

Row/Column Reporting

The alternative to Touch Fields is Row/Column Reporting. When row/column reporting is in effect, you will receive touch reports in the following format:

```
ESC - z  <row>  x  <col>  y  <type>  Q
```

The `<row>` and `<col>` parameters are the 8-bit binary row and column positions of the touch. Possible values for `<type>` are:

3 = Row/Column touch reporting.
4 = Row/Column release reporting.

When you use Row/Column Reporting, you are not required to define any touch fields (remember that the sensing mode parameter, `<mode>`, is part of the touch field definition). The next section explains how to specify row/column reporting mode.

Specifying Reporting Modes

Once you have specified a field, you need to to specify a reporting mode. This is done with the following escape sequence:

```
ESC - z  <smode> n  <tmode> M
```

The meaning of each parameter is:

ESC	The escape character, ASCII 27 decimal.
- z	The sequence which for setting touch sensing mode.
<smode> n	This parameter specifies the screen mode. Use this sequence to determine what type of reporting to perform. Possible values for <smode> are: 0 = Turn off all reporting. This has the effect of turning off reporting without deleting any fields. Softkeys remain touch active. 1 = Enable Row/Column reporting only. This disables field reporting, if active, and enables row/column reporting only. You will receive row/column reports even when a field area is touched. 2 = Enable touch field reporting only. The only reports you will receive will be from defined fields. 3 = Enable both row/column and touch field reporting. Touch fields are reported as defined. Row/Column reports are made from all other areas of the screen.

4 = Toggle touch screen on/off. When off, all touch screen operations are disabled. This disables softkey fields as well.

`<tmode> M`

This parameter specifies the touch mode. It is not required for touch fields, because the sensing mode is set in the field definition. This parameter *is* required for row/column sensing. If used with touch fields, this mode should correspond to the mode specified in the ESC-zg escape sequence.

Valid parameters for `<tmode>` are:

1 = Report on touch only.
2 = Report on release only.
3 = Report on touch and release.

Deleting Touch Fields

You may wish to delete all touch fields or selectively remove one or more fields. For example, if several options are presented as touch fields, you may wish to remove all fields to go on to another menu. At other times, you may only want to make certain fields invalid by selectively deleting those fields you don't want.

The following escape sequence lets you delete the touch field which starts at the row specified by `<row>` and the column specified by `<col>`. If no field starts at the specified coordinates, no action is taken.

```
ESC - z d <row> r <col> C
```

The second form deletes all touch fields. It should be used whenever your application moves from one menu of touch fields to another menu of touch fields. Deleting fields conserves terminal memory.

```
ESC - z D
```

Controlling User-Defined Softkeys

The User-Definable Softkeys are touch sensitive by default. For an explanation on defining them, see the section on "User-Definable Softkeys". If you wish to change the touch sensitive nature of the softkeys, do so with the following escape sequence:

```
ESC - z <key> s <mode> K
```

The parameters are:

ESC	The escape character, ASCII 27 decimal.
- z	The sequence indicating touch screen control.
<key> s	The softkey number. <Key> may be from 1 through 8 inclusive.
<mode> K	The keymode. Values of <mode> are: 0 = Disable touch on <key>. 1 = Enable touch on <key>.

Touch Screen Reset

One final operation remains concerning touch screen, the "reset" operation. Resetting touch screen turns all fields to the off state. Remember, this affects the field state, not touch sensing or reporting. Use this sequence to reset touch fields:

```
ESC - z J
```

This makes most sense with the toggle fields. When a toggle field is reset, no "off" sensing occurs.

Programming Considerations

Touch screen is a very easy way to provide unique features to your application. There are a few things to be aware of when using touch screen.

You probably want to select one type of field and stick with it. ASCII fields are probably the best, because it is easy to "equate" them with valid keyboard input. This means that your user can specify input either through the keyboard or by touch screen. If you use another type of field, you are going to do a lot of "escape sequence parsing" to determine which touch field was affected (touched).

You should also keep touch field responses to a minimum number of bytes. You do this for two reasons:

1. Your application screens will experience a minimum amount of interference from touch input.

2. You conserve the limited terminal memory space. HP has not documented the actual amount of space available for touch buffers, but when you reach the limit, your system hangs up or crashes.

This second point leads to another thing to watch. When you are finished with a touch field, delete it! Also you shouldn't redefine a field over and over in a loop. These two program segments illustrate this point:

```
PROGRAM A

1000 PRINT HOMEUP$;          'Clear screen w/ cursor at top
1010 GOSUB 2000              'Define a touch field: actual
                             ' field not critical
1020 A$=INPUT$(1)            'Read one character from touch
                             'or keyboard
1030 IF A$="1" GOTO 1500     'Go off on option 1
1040 GOTO 1010               'Not option 1, return for more
```

Now look at Program B:

```
PROGRAM B

1000 PRINT HOMEUP$;          'Clear screen w/ cursor at top
1010 GOSUB 2000              'Define a touch field: actual
                             ' field not critical
1020 A$=INPUT$(1)            'Read one character from touch
                             'or keyboard
1030 IF A$="1" GOTO 1500     'Go off on option 1
1040 GOTO 1020               'Not option 1, return for more
```

See the difference? Look at Line 1040. In Program A, the touch field is defined EACH time a character is accepted. In Program B, the touch field is defined only once. Program A will eventually crash the system. Program B will not crash the system as long as it stays within the above loop.

When you use touch screen, the touch fields will "auto-repeat" just as the keys on the keyboard do. To prevent this from confusing your application, use the "touch sensing mode" escape sequence to control acceptance of input. For example, notice how this program controls the touch screen:

```
PROGRAM C

1000 PRINT HOMEUP$;             'Clear screen w/ cursor at top
1010 GOSUB 2000                 'Define a touch field
1015 PRINT CHR$(27);"-z2N";     'Turn on sensing
1020 A$=INPUT$(1)               'Read one character from touch
                                'or keyboard
1025 PRINT CHR$(27);"-z0N";     'Turn off sensing
1030 IF A$="1" GOTO 1500        'Go off on option 1
1040 GOTO 1015                  'Not option 1, return for more
. . .
2000 PRINT CHR$(27);"-zg1,5r1,5c1b1a2m1L1";
2005 RETURN
```

Note lines 1015 and 1025. They turn sensing on just before input is expected, and turn it off immediately after input is received

Of course, at line 1500 in all of the above examples, you should delete the touch field selected. The possible exception might be the case where control is returned to line 1020. to avoid inputting additional (unwanted) characters.

If more than one field is defined over a particular area of the screen, the HP 150 will report only the *most recently* defined buffer. This nature of the system can prove useful.

Sometimes you will want to know if the user is touching the screen but is not touching a defined field. You can do this by defining the entire screen as a field with no enhancements for 'on' or 'off' states. Then define your application fields 'over' this background field. When a user touches one of your fields, you will receive the buffer you expect. If the user touches anywhere else, you will receive the buffer associated with the background field.

CONTROLLING THE KEYBOARD

Some applications on the HP 150 will want to assume more control over the keyboard than required for simple data entry. Some examples of the functions you may want to 'trap' within your application include:

- Cursor Control Keys
- [Next] and [Prev] keys
- Scroll Up and Scroll Down Keys
- Character and Line Manipulation keys

In general, those functions which can be executed by escape sequences can be trapped. Remember, there are some keyboard controls which do not have corresponding escape sequences and which cannot be captured without using AGIOS calls. If these keys and controls must be utilized by your application, refer to the section on AGIOS function calls.

The method used to trap these functions is to cause the terminal to "transmit" the escape sequence associated with the function rather than to "execute" it normally.

For example, in normal operation, typing [Clear line] causes the terminal to erase all of the characters from the current cursor position to the end of line. In "Transmit Function Key" mode, the terminal sends two characters (ESC and K) to your application. The HP 150 does *not* clear to the end of the line unless your application echos the two characters to the display.

The two character escape sequence returned to your application is the same sequence that performs a [Clear line]. The characters returned to your application in Transmit Function Key mode are always the escape sequence characters which correspond to that function.

The terminal executes the function only when it is echoed to the console. For this reason, you will want to use a console input statement in your application language which does not echo to the console. An example of this input statement is contained in the example below.

In the following BASIC program segment, once "transmit function keys" is enabled, the only control key which is executed is the [Clear line] key which generates ESC K. Incidently, this segment assumes that "transmit functions keys" has already been enabled.

```
2000 CH$ = INPUT$(1)          'Read one character w/o echo
2010 IF CH$ <> ESC$ THEN 2070 'Not ESC char
2020 CH$ = INPUT$(1)          'Read the next character
2030 IF CH$ = "K" THEN 2050   'Clear Line
2040 GO TO 2000
2050 PRINT ESC$;"K";
2060 GO TO 2000          'Go get another character
2070 PRINT CH$;          'Have to echo it so it is seen!
```

that this example is not necessarily the best example of good programming techniques. There are efficient ways to code the routine: it is intended to clearly show what is happening.

v To Get Transmit Function Key Mode Started

Transmit Function Key mode is controlled by setting or clearing the "A" Strap in the HP 150 ninal Configuration Menu. This can be done by the user (see the *HP 150 Personal Computer Owner's te* for details) or programmatically as follows:

ESC & k 1 A	Enable Transmit Mode
ESC & k 0 A	Disable Transmit Mode

he above example, you would probably include the following line above the code segment shown.

```
)0 PRINT CHR$(27);"&k1A";
```

rogramming Considerations

sing 'transmit function keys' mode allows you to have additional control over the HP 150. When using is mode, there are some things of which to be aware.

irst, if you set 'transmit function keys' ON when your application starts, be sure to turn it OFF when you xit. This puts the HP 150 in its default state, and assures that your application won't disrupt the inctioning of other applications.

he 'transmit funtion keys' mode causes the terminal to send an escape sequence to your application. The equence you receive is the same sequence which, if sent by your application, would perform the same task s the key. To effectively control the keyboard then, your application should perform input without echo. 3y doing this, your application will receive the two character escape sequence and be able to determine he suitable course of action without the HP 150 having already performed the task.

Finally, remember the User Defined Softkeys have default values which return two character escape sequence. These defaults are:

[f1]	ESC p
[f2]	ESC q
[f3]	ESC r
[f4]	ESC s
[f5]	ESC t
[f6]	ESC u
[f7]	ESC v
[f8]	ESC w

You may use these default values, but you may find it easier to defined the softkeys to fit your application. Finally, there are keys which do not generate escape sequences. For these, you will need to use AGIOS. With AGIOS, you can define all of the special keys to execute normally, to be intercepted by your application, or to be ignored. If you find you need more control than available via the 'transmit function keys' mode, you will need to incorporate AGIOS into your application.

CONTROLLING THE KEYBOARD

The HP 150 supports a number of display enhancements including video enhancements and alternate character sets. You can also selectively 'turn off' the alpha display with or without affecting the softkey labels.

Display Enhancements

Four types of display enhancements are available on the HP 150. You can use them one at a time, or in any combination of the four.

The general format of the escapse sequence used to enable enhanced text is:

```
ESC & d <enh>
```

The possible values for <enh> are provided in the following table:

<enh> :	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Blnk :		*		*		*		*		*		*		*		*
Inv :			*	*			*	*			*	*			*	*
Ulin :					*	*	*	*					*	*	*	*
HBrt :									*	*	*	*	*	*	*	*

Here, the enhancements are Blinking (Blnk), Inverse (Inv), Underlined (Ulin), and Half-Bright (Hbrt).

Once enabled, the enhancement remains on until turned off (with the <enh> of '@') or until the cursor is moved to a different line. In this respect, the HP 150 is different from many other systems.

Character Set Selection

The HP 150 includes a number of different characters sets which can be included in your application. The sets supported are:

Character Set	Description
Base	The system's base set as defined in the configuration menu. This is normally the US ASCII character set.

`Math`	This is a set of mathematics characters such as Greek letters and various symbols.
`Line`	The line drawing set permits the use of various line segments, corners, and symbols.

There are two other character sets which can be accessed only from AGIOS area and line functions: an *Italics* character and a **Bold** character set. Refer to Section 5 for additional information about using those character sets.
The escape sequence for selecting an alternate character set is:

`ESC ) <cset>`

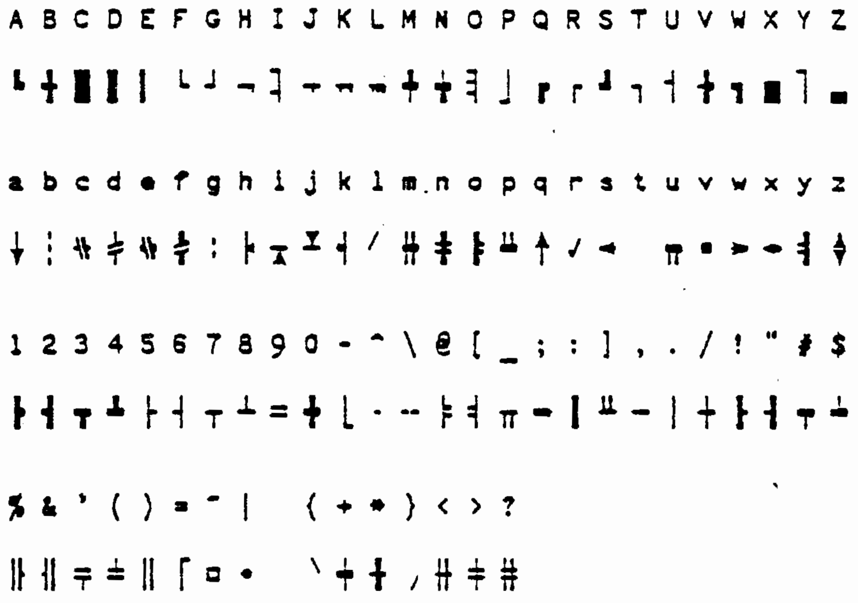
The value for `<cset>`, the character set selection, is:

`@`	:	Base set selected as alternate character set
`A`	:	Math set selected as alternate character set
`B`	:	Line drawing set selected as alternate character set

Once you have selected one of these sets as the alternate set, you can shift between the primary set and the alternate set with the ASCII Shift-Out (SO) and Shift-In (SI) characetrs. The decimal values for these characters is 14 and 15 respectively.

You can use the alternate set by simply printing the ASCII character which corresponds to the desired character using the following table:

<cset>	Character Set Equivalents on the HP 150
Line Drawing Set	


A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
1 2 3 4 5 6 7 8 9 0 - ^ \ @ [_ ; :] , . / ! " # $
% & ' ( ) = ‾ | { + * } < > ?

Math Characters

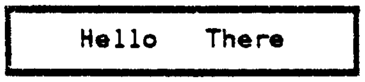
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
∀	∈	⊂	⊃	∃	∉	⊆	ℵ	∩	⋂	⊕	¬	∧	≠	•
P	Q	R	S	T	U	V	W	X	Y	Z	a	b	c	d
℘	∅	⊇	∋	━	∪	∨	⋃	×	⌀	■	α	β	ψ	φ
e	f	g	h	i	j	k	l	m	n	o	p	q	r	s
ε	∂	λ	η	ι	θ	κ	ω	μ	ν	ρ	π	γ	ϑ	σ
t	u	v	w	x	y	z	1	2	3	4	5	6	7	8
τ	ξ	Δ	δ	×	∪	ζ	^{1}	^{2}	^{3}	^{4}	^{5}	^{6}	^{7}	^{8}
9	0	-	^	@	[	_	;	:	]	,	.	/	!	"
^{9}	^{0}	≡	←	⊖	⟨	↧	Λ	Ω	⟩	Ψ	Φ	Ξ	√	|
#	$	%	&	'	(	)	=	‾	|		{	+	*	}
§	∇	±	∝	⌠	+	=	⌡	←	→		↑	Γ	Π	T
<	>	?												
∞	↑	Σ												

By using the above table, you can write program which draw boxes and lines, use special math symbols, and even use enhancements on these special characters, all from within BASIC. Here is a sample of such a program which draws a rectabgle on the screen:

```
2500 'Print a box with 'Hello There' in the middle
2510 PRINT CHR$(27);")B";      'Select Line Drawing as Alternate
2520 PRINT CHR$(14);"Q;;;;;;;;;;;;;;;;;;;;;;W"   'Top line
2525 ' Now print middle line with normal characters in middle
2530 PRINT CHR$(14);":";CHR$(13);"    Hello   There     ";
2540 PRINT CHR$(14);":"          'Other end of box
2545 ' That concludes the middle line
2550 PRINT CHR$(14);"A;;;;;;;;;;;;;;;;;;;;;;S"  'Bottom of box
```

Note that, to print the normal characters in the middle, you need to shift back to the primary character set (lines 2525 through 2545).

The above program will produce a figure which looks like this:



```
Hello   There
```

Alphanumeric Memory Control

The alphanumeric screen and graphics screen images are independent. For this reason, the two can be displayed together, or either can be displayed alone. As you will see, there is a 'Graphics Text' mode which writes alpha characters into graphics memory, and that is controlled totally within graphics.

In the alpha memory, you can turn off the entire display including softkeys or you can turn off any text and leave the softkey labels intact. To perform these tasks, use the following escape sequences:

```
ESC & w 12 F     Alpha Display On
ESC & w 13 F     Alpha Display Off (softkey labels remain)
ESC * d E        Alpha Memory On
ESC * d F        Alpha Memory Off (including softkey labels)
```

You will see more on these final two sequences in the section on display control later in this section.

Programming Considerations

In using both display enhancements and alternate character sets, keep in mind how the HP 150 works. When the display is cleared, none of alpha memory is allocated. When you position the cursor and turn on a display enhancement, that enhancement affects all bytes on that line which may be allocated now or in the future.

For example, on a clear screen position the cursor to column 5 on the first row. Print the 'inverse video' enhancement, ESC & d B. Now print several characters: they will appear in inverse text. If your application leaves the first line and returns to column 70 (for example) on the first line, the next characters you print will cause the entire line to become inverse. This occurs ***even if you do not include the 'inverse video' sequence at column 70!***

To avoid this situation, you should turn on inverse video; print the text to be inverse; and turn off inverse video with the ESC & d @ sequence. Then, repositioning the cursor to the right on the same line will not cause the entire line to be enhanced.

Alternate characters work the same way. Once you have shifted out (SO) to an alternate set, all characters to the right will be shifted unless and until you print a shift-in character (SI). The example above illustrated this point: be sure you understand how it works.

ESCAPE SEQUENCE REFERENCE

Introduction

An escape code lets you execute terminal operations from a program. When your HP 150 receives an escape code from an executing program, it performs the operation specified in the code just as if it had been entered by an operator at the keyboard. The operations performed can range from controlling the appearance of the display screen, such as displaying text bright or half bright, to re-defining a function key to be used by an application program you are writing.

All escape codes begin with the escape character, ESC, (ASCII 27) followed by an one or more ASCII characters that make up the body of the escape code. This combination of the ESC character and the following code form an escape sequence.

Most escape sequences can be performed by entering them from the keyboard. However, if the HP 150 is in Remote Mode, you will find most operating systems (including MS-DOS) will perform special processing on the ESC character and will not permit you to enter escape sequences. Take the terminal out of Remote Mode to experiment with sequences.

Some escape sequences are made up of two characters, and some are made up of more than two characters, or multiple characters. Two-character and multiple-character escape sequences perform different types of functions, and the rules for entering them differ.

Two-character escape sequences generally correspond to functions performed by a single keystroke at they keyboard. For example, the horizontal TAB key at the left of your terminal keyboard corresponds to the two-character escape sequence

```
ESC I
```

which performs a TAB ("forward tab"). In two-character escape sequences, the escape code character must be specified exactly. The following sequence

```
ESC i
```

using the lower-case "i" performs a BACKTAB. Note that, in general, upper case and lower case parameters have very different meanings to the system.

Multiple-character escape sequences generally refer to functions that require one or more parameters. Placing the cursor to a particular column and row position on the screen requires a multiple-character escape sequence. For example, the sequence

```
ESC & a 5 c 1 0 R
```

places the cursor at the fifth column of the tenth row on the terminal screen. In this case, however, the sequence

```
ESC & a 1 0 r 5 C
```

performs exactly the same function. While the order of the parameters in an escape sequence is not important, the case is significant.

Because the number of characters in a given sequence depends upon the <row> and <column> numbers, and because several parameters may be combined in one escape sequence, the terminal must have a way of knowing where a sequence terminates. For this reason, the final parameter in a multiple escape sequence must be an upper case letter, and all preceding parameters must be lower case. **The first upper case letter following the ESC character terminates the sequence.**

Multiple-character escape sequences of the same type may be combined into a single sequence. Sequences fall into five major groupings:

ESC & <parameters>
ESC * <parameters>
ESC) <parameters>
ESC (<parameters

These groupings may be broken down further by additional characters as in:

ESC & k
ESC & s

and so on.

For example, if you wished to change the values of the Local Echo and Caps Lock parameters in your terminal configuration menu, you could change individual values using the following sequences:

Value:	Escape Sequence:
Local Echo = yes	ESC & k 0 L
Local Echo = yes	ESC & k 1 L
Caps Lock = no	ESC & k 0 C
Caps Lock = yes	ESC & k 1 C

To change both values, or any parameters within the ESC&k group, you could use one sequence. For example, to set Local Echo = Yes and Caps Lock = Yes, you could use either of the following sequences:

ESC & k 1 1 1 C
or
ESC & k 1 c 1 L

Note that the final identifier must be upper-case, and the preceeding identifiers must be lower-case.

NOTE

For the purpose of these and other examples, the elements of the escape sequences have been separated with spaces. When you enter escape sequences at the terminal, or print them from a program, however, all characters must be contiguous, as in this sequence:

```
ESC & a 5 c 1 0 R
```

Parameters for escape sequences are enclosed in angle brackets (<>).

Block Transfers from the HP 150 Terminal to a Remote Host

The HP 150 normally operates in character mode, sending each character to the host as the key is pressed. It is also capable of operating in Block Mode, which requires special handshking with the host. 'Host' here indicates a remote system, or the MS-DOS operating system.

Even in character mode, certain transfers occur in Block Mode. These operations will result when:

- The ENTER key is pressed.
- A transfer is initiated by ESC d (ENTER Sequence)
- Data transfered when Line Modify or Modify All Mode are on
- Transfer of a "User-Defined Softkey" function key when it is pressed in Remote Mode.
- Status data transfers such as cursor position or primary status

The actual protocol of the data transfer depends on the settings of the following configuration switches:

Terminal Mode:	Default Setting:
Line/Page(D)	Line
InHndShk(G)	No
Inh DC2(h)	No
AutoTerm(J)	No
ClearTerm(K)	No

Several other parameters affect block mode transfer including the state of Block/Character Mode, Line Modify/Modify All Modes, Format Mode, and Auto Linfeed.

HANDSHAKING.

The InhHndShk(G) (Inhibit Handshake, strap G), and Inh DC2(H) (Inhibit DC2, strap H) selections on the Terminal Configuration Menu determine, in general, the type of transfer which will occur. The possible types of transfer, or handshakes, can be configured:

1. No handshake, also known as a Type 1, in which the terminal simply sends the data block without any special control from the host system. This occurs with 'G' and 'H' set.

2. A DC1 Trigger handshake (Type2), in which the host computer must trigger the block transfer with a DC1 character. This occurs with 'g' and 'h' set.

3. A DC1/DC2/DC1 handshake (Type 3). Here, a DC1 from the host is answered with a DC2 from the terminal. The host must respond with a second DC1. When this occurs, the terminal will transmit the data. This occurs with 'g' and 'H' set.

DISABLE AUTO LINE FEED.

When Auto Line Feed is enabled, all block transfers are preceded by a line feed character. This can cause misalignment of the cursor before the transfer begins, a can also cause difficulties whild reading the data transmitted. For these reasons, disabling Auto Line Feed is recommended.

CORRECT HANDSHAKING.

MS-DOS and its applications do not generally provide the DC1 handshake expected by the HP 150 terminal with the 'g' and 'h' default settings. Because the HP 150 keyboard is buffered, you are probably safe in disabling any handshaking by setting the two straps to 'G' and 'H' with the following escape sequence:

```
ESC & s 1 g 1 H
```

This sets both straps so that no handshaking occurs during Block Mode data transfers. Note that this may not be suitable for connection to a remote computer where data may be lost because of data over-run.

In such a case, the remote system would have to generate the appropriate DC1 trigger character. Incidently, the HP 3000 performs this handshaking independent of the applicaition, so these notes apply only to non-HP 3000 mainframes.

A sample program follows which illustrates how a remote host might generate the handshaking required for block transfers. Note this program will work properly on the HP 150 as well. If you do not disable the 'G' and 'H' straps, this program would be required on the HP 150 as well.

```
1000 REM Read one line from the display
1010 PRINT CHR$(27);"d";          'Programmatic ENTER
1020 PRINT CHR$(17);              'DC1 Trigger
1030 LINE INPUT TEXT$             'Read data
```

CORRECT HANDSHAKING.

The areas where your local application will be affected by block transfers will be in three areas:

1. Programmatic ESC d to read a line of the screen simulating an ENTER

2. A terminal status request

3. User Defined Softkeys

To handle the first case, a programmatic read of the screen, the method illustrated above is fine. If you do disable the 'G' and 'H' straps, a simpler routine would be"

```
1000 PRINT CHR$(27);"d"; 1020 LINE INPUT"",BUF$
```

If the length of the expected data is known, you can accept just the number of bytes you expect. The following example shows how you might read the current cursor position from Microsoft BASIC.

```
1000 REM Get the current alpha cursor position
1010 PRINT CHR$(27);"a";
1020 BUF$=INPUT$(12)
```

You must accept 12 characters because the reported cursor position is sent in the form:

```
ESC & a nnn c mmm Y <CR>
```

Where 'nnn' is the column number and 'mmm' is the row number. Note that printing this same sequence would re-position the cursor to its current location.

Using the Keyboard and Display

There are certain two-character escape sequences which perform keyboard functions. These functions all affect the way in which information is displayed at your terminal.

Refer to the *HP 150 Terminal User's Guide* for the corresponding keystrokes.

CURSOR CONTROL OPERATIONS.

The display portion of the terminal consists of display memory and the display screen. Display memory is the portion of terminal memory assigned to contain alphanumeric data entered to the terminal or display on the screen; it consists of 48 rows numbered 0 - 47. The display screen is the visible part of display memory and consists of 27 rows; each row contains space for 80 characters numbered 0 - 79. The first 24 rows of the display screen are used to display one "page" of display memory; rows 25 and 26 display the function key labels (softkeys), and row 27 contains information about the terminal's status, and the current date and time.

As data is added to display memory, lines are moved off the top of the display screen; when display memory is filled, lines are lost out of display memory. The status line indicates the number of lines entered into display memory.

The cursor is represented on the screen by a blinking underscore mark or a blinking box in inverse video (green on a black background), and indicates where the <u>next</u> character entered will appear. The cursor only appears on the screen display.

MEMORY ADDRESSING.

Display memory can be addressed using absolute or relative coordinate values. The types of addressing are:

- Absolute
- Screen Relative
- Cursor Relative

In absolute addressing mode, you specify the row and column within display memory. Column values range from 0 to 79, while row values range from 0 to 47.

In screen relative mode, you specify the position relative to the 'window' where the first line on the screen is row 0. If scrolling has occurred, screen relative row 0 may be absolute row 10, for example.

In cursor relative mode, you specify a position relative to the current cursor position by specifying signed row and column values. A plus sign indicates movement to the right or down, while a negative sign indicates movement to the left or up.

The cursor control operations are as follows:

Absolute Cursor Positioning

ESC & a <col> c <row> R

Screen Relative Addressing

ESC & a <col> c <row> Y

Cursor Relative Addressing

ESC & a <scol> c <srow> Y

In the above examples, <col> and <row> represent numeric ASCII characters for the column and row locations respectively. The <scol> and <srow> represent signed numeric ASCII.

You may use a combination of display screen, display memory, and cursor-relative coordinates within a single escape sequence.

Examples:

To position the cursor at column 70, 18 rows below the current cursor location, use the following sequence. Note that text will scroll 'up' if necessary to reach that line.

ESC & a 6 9 c + 1 8 R

To move the cursor so that it is positioned at the character 15 columns to the left of the current cursor position in the 4th row currently visible on the display screen, enter:

ESC & a - 1 5 c 3 Y

To position the cursor to column 5 on the current row, use:

ESC & a 5 C

Programming Considerations

The one thing to be aware of in cursor positioning is the effect of memory lock. When memory lock is enabled, 'freezing' part of the text at the top of the screen, absolute memory addressing does not function properly. To be on the safe side, always use screen relative addressing.

If scrolling is something you don't wish to allow, you can effectively disable it by using 'Transmit Function Keys' mode in combination with 'home up' commands whenever you paint a form on the screen. This will allow you to capture the scrolling functions, and be certain your addressing is always correct.

Mode Selections

Escape sequences can be used to change the active values of the mode function keys and terminal configuration parameters listed in the table below. A change of a parameter value using one of the sequences listed below takes effect immediately, but the content of non-volatile memory is not changed. If a configuration menu is displayed on the screen when the escape sequence is received, the sequence is not executed until you exit from the menu.

Using an escape sequence, you can "lock" the current configuration menus so that the menu can not be altered from the keyboard or from a program. Any attempt to access a locked menu will result in a "beep" from the bell and an error message. Note that when the configuration menus are locked, the MODIFY ALL, BLOCK MODE, REMOTE MODE, and AUTO LF mode function keys are also locked.

To unlock the menus, use the following escape sequence:

ESC &q 0L

To lock the menus, use the following escape sequence:

ESC &q 1L

The following escape sequences select active values without changing the values in non-volatile memory.

Entries in the MENU FIELD/MODE column that are marked with an asterisk are represented on the Terminal Configuration Menu.

ESCAPE SEQUENCE	MENU FIELD/ MODE	ENTRY VALUE	x
ESC &k <x>A	AUTO LF	OFF	x=0
		ON	x=1
ESC &k <x>B	BLOCK MODE	OFF	x=0
		ON	x=1
ESC &k <x>C	*Caps Lock	OFF	x=0
		ON	x=1
ESC &k <x>D	*Bell	OFF	x=0
		ON	x=1
ESC &k <x>I	*ASCII 8 Bits	NO	x=0
		YES	x=1

ESCAPE SEQUENCE	MENU FIELD/ MODE	ENTRY VALUE	x
ESC &k <x>K	Auto Keyboard	OFF	x=0
	Lock Mode	ON	x=1
ESC &k <x>L	*LocalEcho	OFF	x=0
		ON	x=1
ESC &k <x>M	MODIFY ALL	OFF	x=0
		ON	x=1
ESC &k <x>N	SPOW(B)	OFF	x=0
		ON	x=1
ESC &k <x>O	Numeric pad	---	x=0
	Graphics pad	---	x=1
ESC &k <x>P	Caps Mode	OFF	x=0
		ON	x=1
ESC &k <x>Q	Click	OFF	x=0
		ON	x=1
ESC &k <x>R	REMOTE MODE	OFF	x=0
		ON	x=1
ESC &s <x>A	*XmitFnctn(A)	NO	x=0
		YES	x=1
ESC &s <x>B	*SPOW(B)	NO	x=0
		YES	x=1
ESC &s <x>C	*InhEolWrp(C)	NO	x=0
		YES	x=1
ESC &s <x>D	*Line/Page(D)	LINE	x=0
		PAGE	x=1
ESC &s <x>G	*InhHndShk(G)	NO	x=0
		YES	x=1
ESC &s <x>H	*Inh DC2(H)	NO	x=0
		YES	x=1
ESC &s <x>J	*Auto Term(J)	NO	x=0
		YES	x=1

ESCAPE SEQUENCE	MENU FIELD/ MODE	ENTRY VALUE	x
ESC &s <x>K	*ClearTerm(K)	NO	x=0
		YES	x=1
ESC &s <x>L	*InhSlfTst(L)	NO	x=0
		YES	x=1
ESC &s <x>N	*Esc Xfer(N)	NO	x=0
		YES	x=1
ESC &s <x>W	*InhDcTst(W)	NO	x=0
		YES	x=1
ESC &x <x>C	Send Cursor Position mode	OFF	x=0
		ON	x=1

Programming Considerations

When your application first executes, you will want to send a 'set-up string' to the HP 150 to assure the state of all the straps and modes which can cause trouble.

The following sequence is a good starting point for all applications:

ESC & k	0 a	Auto Line Feed off
	0 b	Block Mode off
	0 c	Caps Lock off
	1 d	Bell on
	0 i	ASCII 7 bit mode
	0 k	Keyboard Lock off
	0 l	Local Echo off (done by MS-DOS)
	0 m	Modify All Mode off
	0 n	Space Overwrite Mode off
	0 o	Numeric Keypad on
	0 p	Caps Mode off
	1 Q	Keyclick on

You will also want to use:

ESC & s	0 a	Transmit Function Keys off *
	0 b	Space Overwrite Disabled
	0 c	Inhibit end of line wrap-around
	0 d	Line Mode Enable
	1 g	Inhibit Block Mode Handshaking
	1 h	Inhibit DC2 Trigger
	0 j	Auto Terminator Disabled
	0 k	Clear Terminator Mode off
	1 l	Inhibit Self Test
	0 n	Escape Sequence Transmit to printer disabled
	1 W	Inhibit DataComm Test

* If you have decided to use 'Transmit Function Keys' mode, this will be a 1 A.

Format Mode

DEFINING FIELDS.

From a program executing in a host computer, you can define "unprotected" and "transmit-only" fields with the various screen attributes by using the following escape sequences:

ESC [Start	s an unprotected field.
ESC { Start	s a transmit-only field.
ESC] Ends	a field.
ESC & k <x>Z	The data is transmitted when the ENTER key is pressed depending on these values for <x>:
0 =	Transmits data within the Unprotected and Transmit-Only fields (default).
1 =	Transmits data from Transmit-Only fields, and any unprotected fields which have been modified.

The two-character escape sequences ESC 6, ESC 7, and ESC 8 are used to specify editing which can be done by the HP 150 as the user enters data. The meanings of these controls follows:

ESC 6	Specifies alphabetic field only. No numeric or special characters are accepted
ESC 7	Specifies numeric only field. No alpha or special characters are accepted
ESC 8	Specifies unrestriscted characters field. Any character may be entered within this field.

Notice you use the same sequence of operations and subfields programmatically as when doing so through the keyboard. For example, if you wish a field to include video enhancements, you must issue the appropriate ESC & d sequence before issuing the ESC [or ESC { sequence. To define the start of a subfield, you must first issue the appropriate display enhancement sequence (ESC 6, ESC 7, or ESC 8) at the point where the subfield is to begin.

Forms are defined with Format Mode disabled, but are not interpreted as such until Format Mode is initiated. When Format Mode is initiated, all of display memory is "protected" except for portions which have been explicitly defined as "unprotected" and "transmit-only" fields.

You can enable and disable Format Mode programatically using the following escape sequences:

```
Enable  : ESC W
Disable : ESC X
```

HOW TO TRANSFER FORMS FROM THE SCREEN TO A HOST COMPUTER.

When writing application programs that will display a form structure on the terminal screen, you may choose to code the program statements that issue the necessary escape sequences, SO and SI codes, and data. For complex form structures however, this method can be both tedious and prone to error.

An easier method is to design the form at the terminal and then transfer the form structure from the screen to the host computer where it can be accessed by or incorporated into your program. If the terminal is connected to an HP 3000 Computer System, you may use the FORMSPEC portion of V/3000, and then include appropriate V/3000 intrinsic calls in your application programs to use the form in the run-time environment.

Graphics Display Control

The following escape sequence controls the graphics display.

ESC *d <z>	Performs the action indicated by <z> on the graphics display. Values for <z> are: a = Clear graphics memory b = Set graphics memory c = Turn on graphics display d = Turn off graphics display e = Turn on alphanumeric display f = Turn off alphanumeric display k = Turn on graphics cursor l = Turn off graphics cursor m = Turn on rubber band line n = Turn off rubber band line q = Turn on alphanumeric cursor r = Turn off alphanumeric cursor s = Turn on graphics text mode t = Turn off graphics text mode z = No operation
ESC *d <x,y> o	Move graphics cursor to horizontal position <x> and vertical position <y> (relative to the origin).
ESC *d <x,y> p	Move graphics cursor to horizontal position <x> and vertical position <y> (relative to its present location).

GRAPHICS LABEL TRANSMISSION.

This escape sequence is used for transmission of a graphics text label from a program to the terminal:

ESC *l <text> CR, CR LF, LF CR, or LF	The characters contained in <text> are printed on the display starting at the current pen position.

VECTOR DRAWING.

The following escape sequences are used to draw vectors.

ESC *m <x>a	Selects the drawing mode specified by <x>. The values for <x> are: 0 = No effect 1 = Clear (turn off graphics bits) 2 = Set (turn on graphics bits) 3 = Complement (toggle the graphics bits) 4 = Jam (turn bits on or off according to the data)
ESC *m <x>b	Selects the line type specified by <x>. The values for <x> are: 1 = Solid line 2 = User line pattern 3 = Current area pattern 4-7 = Lines 4-7, respectively 11 = Point plot
ESC *m <x> <y>c	Defines an eight-bit segment of line pattern and a scale where: <x> is a number from 0 to 255 which, when converted to its binary form, illustrates the segment of line pattern. <y> is a number from 0 to 255 which indicates the number of times the line pattern should be repeated.
ESC *m <a b c d e f g h>d	Defines an 8 x 8 pattern where <a> through <h> are numbers from 0 through 255 which, when converted to their binary values and stacked, illustrate the pattern.
ESC *m <x1,y1, x2,y2,>e	Defines a rectangular area to be filled, where <x1>, <y1> and <x2,y2> define the rectangle located with respect to the absolute origin.
ESC *m <x1,y1, x2,y2>f	Defines a rectangular area to be filled, where <x1>, y1> and <x2,y2> define the rectangle with respect to the relocatable origin.
ESC *m <x,y>j	Locates the relocatable origin at coordinates <x,y> with respect to the absolute origin.

ESC *m <x>g

Selects an area pattern specified by <x>. The values for <x> are:

1 = Solid area fill
2 = User-defined area fill (default)
3 = Predefined pattern 0 (short dashed hatching)
4 = Predefined pattern 1 (long dashed hatching)
5 = Predefined pattern 2 (hatching)
6 = Predefined pattern 3 (cross hatching)
7 = Predefined pattern 4 (fine cross hatching)
8 = Predefined pattern 5 (medium checker-board)
9 = Predefined pattern 6 (fine checker-board, 1:1 blend)
10 = Predefined pattern 7 (3:1 blend)

ESC *m <x>h

Set area boundary pen <x>; where <x> is an integer in the range -32767 through 32767. The three low bits of the binary form of the integer is used to select the pen (0...7).

ESC *m k

Locates the relocatable origin at the current pen position.

ESC *m l

Locates the relocatable origin at the graphics cursor position.

ESC *m <x>m

Sets the graphics text size to <x>, where <x> is a number from 1 to 8.

ESC *m <x>n

Sets the graphics text orientation to <x>. The values for <x> are:

1 = 0 degrees rotation
2 = 90 degrees rotation
3 = 180 degrees rotation
4 = 270 degrees rotation

ESC *m o

Turns on text slant.

ESC *m p

Turns off text slant.

ESC *m <x>q

Sets the origin of graphics text at location <x> on the display. The values for <x> are:

0 = left/baseline
1 = left/bottom
2 = left/middle
3 = left/top
4 = center/bottom
5 = center/middle
6 = center/top
7 = right/bottom
8 = right/middle
9 = right/top

ESC *m r — Set graphics defaults. The following list gives the default settings that are put into effect when this escape sequence is used:

Pen Condition = Down
Line Type = 1 (solid)
Drawing Mode = 2 (JAM 1)
User-Defined Line Pattern = 255, 1
Area Fill Type = 2 (User-Defined Pattern)
User-Defined Area Fill Pattern= 255, 255,..., (Solid)
Background Pen = 0 (Black)
Primary Pen = 7 (White)
Secondary Pen = 0 (Black)
Boundary Pen = Off
Graphics Text = Off
Text Size = 1
Text Direction = 1
Text Origin = 1 (left, bottom)
Text Slant = Off
Text Color = Primary Pen
Relocatable Origin = 0,0
Alpha Video = On
Graphics Video = On
Alpha Cursor = On
Graphics Cursor = Off
Graphics Cursor Address = 0,0
Rubberband Line = Off
Compatibility Mode:
    Page Full Straps = 0 (Out)
    GIN Strap = 0 (CR Only)

NOTE

Parameters marked with an asterisk are those affected by the sequence "ESC *m <1>r".

ESC *m <1>r — Sets the graphics defaults which are marked with an asterisk in the list above.

ESC *m z — No operation.

PLOTTING COMMANDS.

This escape sequence is used in plotting vectors.

ESC *p <x>	Performs the action specified by <x>. The values for <x> are:

a = Lift the pen
b = Lower the pen
c = Use graphics cursor position as new point
d = Draw a point at the current pen position and lift the pen
e = Set relocatable origin at the current pen position
f = Data is ASCII absolute
g = Data is ASCII incremental
h = Data is ASCII relocatable
i = Data is absolute
j = Data is short incremental
k = Data is incremental
l = Data is relocatable
s = Start area fill
t = End area fill
u = Lift area boundary pen
v = Lower area boundary pen
z = No operation

GRAPHICS STATUS.

This escape sequence reads the graphics status.

ESC *s <x>^	Reads the status type specified by <x>. The values for <x> are:

1 = Terminal I.D.
2 = Pen position
3 = Graphics cursor position
4 = Read cursor position and wait for key
5 = Display size
6 = Graphics capabilities
7 = Graphics text status
8 = Read zoom status
9 = Relocatable origin
10 = Reset status
11 = Area shading
12 = Dynamics

COMPATIBILITY MODE.

These escape sequences are used in Compatibility mode.

ESC *t <x>a	Selects the graphics terminator specified by <x>. The values for <x> are: 0 = CR terminator 1 = CR EOT terminator 2 = No terminator
ESC *t <x>b	Sets and clears Page Full Break strap. The values for <x> are: 0 = Clear 1 = Set
ESC *t <x>c	Sets and clears Page Full Busy strap. The values for <x> are: 0 = Clear 1 = Set
ESC *t <x>d	Sets and clears 4014 mode. The values for <x> are: 0 = 4010 mode 1 = 4014 mode
ESC *t z	No operation.
ESC *w r	Graphics hard reset.

THIS PAGE SHOULD BE BLANK

AGIOS PROGRAMMING

SECTION 5

AGIOS (Alpha Graphics Input/Output Subsystem) is a facility that lets you use system routines to perform tasks on the HP 150 keyboard and display. They let you perform text and graphics mode operations on your display, let you define and use softkeys (function keys), and let you perform all touch screen operations.

AGIOS functions are a unique feature of the HP 150. Comparable facilities on other micro-computers are generally not available. Most micros require that you use hardware specific code in order to make your application run as fast as possible. This means that your program is dependent on specific hardware addresses and revisions of the computer firmware (ROM) and operating system. If the manufacturer changes some small detail in the system, your application may not run. This means that you may spend additional amounts of time and money updating your application, and trying to salvage the goodwill of your customers.

Hewlett-Packard is committed to maintaining AGIOS compatibility in future systems in the Series 150 line. This means that your software, optimized for the HP 150 (using AGIOS function calls and/or escape sequences), should run with no additional modifications on future products like the HP 150. Because of this commitment to future compatibility, HP strongly encourages you to use these facilities when writing application programs. Additionally, because these features are available, we do not document hardware specific details which you might be required to know on other micro-computers.

HOW TO USE AGIOS FUNCTION CALLS

You make AGIOS function calls by using the standard MS-DOS calling conventions. In fact, an AGIOS call is an "I/O Control Write" on the console device (MS-DOS Function 44 hex). In general, each type of AGIOS call requires a different buffer. To use AGIOS function calls, you need to build the appropriate buffer, set up the 8088 registers, then call MS-DOS (with an INT 21H).

The following example shows you how you use AGIOS function calls in assembly language. The function call that is used is the "Execute Two Character Sequence" AGIOS function call, which clears the display screen (refer to this function call in this section). This AGIOS function call requires a three byte buffer which contains the function code followed by the "J" parameter.

```
             ---------------------------
BUFF         | 16   |   0   |    J    |
             ---------------------------
Byte:           + 0     + 1      + 2
```

The program segment to accomplish the clear display would look something like this in assembler:

```
CLRSCRN   MOV    AX,4403H          ;I/O Control Write
          MOV    BX,1              ;Console Handle, always  = 1
          MOV    CX,3              ;BUFF Length - 3 in this case
          MOV    DX,OFFSET BUFF    ;and Offset
          INT    21H               ;MS-DOS call
          RET
BUFF      DB     16,0,'J'
```

When you perform AGIOS function calls from a high level language such as Pascal, you make a call to an assembly language subroutine. You can write the subroutine to perform a particular function call, as in the above example. A more useful subroutine, however, would be a generalized routine to execute any function call, with the parameters being passed when the call is made.

The ISV Distribution Disc contains examples of subroutines callable from MS-Pascal and Lattice C. In addition, BPL BASIC documentation lists a subroutine callable from Basic, and Appendix B in this booklet gives you a detailed example of making AGIOS calls from a Pascal program. If these examples do not include your particular language, they can provide a guide to coding the calls in the language that you're using.

ERRORS DURING FUNCTION CALLS

If an MS-DOS error occurs when you make an AGIOS function call, the "carry flag" is set according to the standard MS-DOS procedures. If an AGIOS error occurs, the *AX* register contains a non-zero value. AGIOS errors can occur when MS-DOS errors do not occur. This means that the carry flag is not set but the *AX* register contains a non-zero character. Therefore, you need to check both of these indicators after a function call to determine if the operation was completed successfully.

SYNTAX USED IN THE AGIOS FUNCTION CALLS

Each AGIOS function call is explained in detail on the succeeding pages. In order to clarify the information that you need to supply, a standard notation is used. Parentheses and positional notation is used as follows:

PARM	Indicates a single byte parameter.
(PARM1,PARM2)	Indicates two single byte parameters with parm1 in the high byte and parm2 in the low byte.
(,PARM)	Indicates a single byte parameter in the low order byte of the word. The high byte is ignored.
(PARM,)	Indicates a single byte parameter in the high order byte of the word. The low byte is ignored.
(PARM)	Indicates a word (16 bit) parameter.
((PARM))	Indicates a double word parameter. Usually the first word is a data segment address and the second word is an offset address.

Where applicable, the AGIOS call name is followed by the corresponding escape sequence. For example, one entry later in this section is:

```
DEFINE TOUCH FIELD (ESC - z g)
```

This indicates that the escape sequence which corresponds to the Define Touch Field AGIOS call is ESC - z g. Additional parameters may be required: refer to Section 4 for exact syntax.

AGIOS FUNCTION CALL REFERENCE

Batch Function Call

A special function call is available which lets you execute a sequence of function calls automatically. Using this function call is especially convenient when you frequently perform the same set of AGIOS function calls.

To "batch" function calls, you set up the sequence of AGIOS function calls in a *command buffer*. Then you issue the following batch function call using the command buffer as one of its parameters.

(0, 0)	Function code
(BUFFER LENGTH)	COMMAND BUFFER length (byte count).
((COMMAND BUFFER))	A pointer to the buffer containing the AGIOS function calls. The function calls are defined consecutively. Use the same parameter format as specified in this section for the individual function calls.

A batch call is aborted when any of the function calls in the batch causes an error condition. Additionally, you cannot nest batch function calls (include them in the batch).

Example:

This example clears the alpha display by homeing up first, then clearing the display. Refer to the "H" and "J" options of the "Execute Two Character Sequence" function call.

The command buffer looks like this:

```
                 -------------------------------
Contents-->      | 16 |  0 |  H | 16 |  0 |  J |
                 -------------------------------
Byte     -->       +0   +1   +2   +3   +4   +5
```

The assembler routine that sets up the command buffer and executes a batch call might look like this:

```
        CLS     PUSH DS                     ;  Save DS on Stack
                POP  CMDSEG                 ;  And Store it in Batch Buffer
;
                MOV  AX,4403H               ;  I/O Control Write
                MOV  BX,1                   ;  Console Handle
                MOV  CX,8                   ;  Batch Buffer Length
                MOV  DX,offset BATCH        ;  Batch Command Buffer
                INT  21H
                RET
;
        CMDBUF  DB   16,0,'H',16,0,'J'
;
        BATCH   DB   0,0                    ; AGIOS Batch Command
        BUFLEN  DW   6                      ; CMDBUF Len 6 Here
        CMDOFF  DW   CMDBUF                 ; CMDOFF equates CMDBUF
        CMDSEG  DW   0                      ; Data Segment Dummy Value
```

Video Intrinsics

The video intrinsics are a set of functions that may be used to update the state of the display. With the exception of the Write Line function, all intrinsics operate on a pre-defined subset area of the 24 by 80 character display. All row and column values are relative to zero. The upper left corner of the display is (0,0) and the lower right corner is (23,79).

A null data buffer pointer (segment = 0FFFFH) will suppress the update operation for that data type. There is a one to one correspondence between the position of a data byte in its buffer and the character position that it will affect in the pre-defined update area, starting at the upper left corner, incrementing column position first and then row position.

The ASCII data consists of the 8 bit HP Standard ASCII character code. The character set data consists of character set code characters as follows:

CHARACTER CODE:	CHARACTER SET SELECTED:
@	Normal Roman
A	Line Drawing
B	Bold Face Roman
C	Italic Roman
D	Math
SPACE	No Change

The enhancement data consists of enhancement code characters as follows:

security off:	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
security on :	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	-
half-bright									x	x	x	x	x	x	x	x
underline					x	x	x	x					x	x	x	x
inverse video			x	x			x	x			x	x			x	x
blinking		x		x		x		x		x		x		x		x

(A space, 20H, indicates no change to the current state.)

DEFINE AREA.

This function specifies the area to be operated upon by subsequent area update operations.

(0, 1)	Function Code.
(LR-ROW,LR-COL)	Defines the lower right corner of the area to be operated upon.
(UL-ROW,UL-COL)	Defines the upper left corner of the area to be operated upon.
((PREV-COORD))	A pointer to a buffer where the previous area coordinates are returned. The format of the returned data is the same as the two input coordinates. This buffer pointer may be null.

WRITE AREA.

This function writes data into the pre-defined display area.

(0, 2)	Function Code
(DATA LENGTH)	Length of the data buffers.
((ENH POINTER))	A double word pointer to a buffer of enhancement code characters to be used to change the enhancement state of the update area.
((CHAR SET))	A double word pointer to a buffer of character set code characters to be used to change the character set state of the update area.
((ASCII POINTER))	A double word pointer to the buffer of ASCII data to be written into the update area.

CLEAR AREA

This function clears the most recently defined area to ASCII blanks (20H), with no enhancements set.

(0, 3)	Function Code

ENHANCE AREA

This function sets the enhancement state of the entire pre- defined display area to the specified enhancement.

(0, 4)	Function Code
(,ENHANCEMENT)	An enhancement code character.

READ AREA

This function reads data from the pre-defined display area.

(0, 5)	Function Code
(DATA LENGTH)	Length of the data buffers.
((ENH POINTER))	A double word pointer to the buffer that the enhancement data will be read into.
((CHAR SET))	A double word pointer to a buffer that the character set data will be read into.
((ASCII POINTER))	A double word pointer to the buffer that the ASCII data will be read into.

SHIFT AREA

This function shifts the data in the pre-defined display area. Enhancements and character set are shifted with the ASCII data. Data shifted off an edge of the update area is lost.

(0, 6)	Function Code
(DATA LENGTH)	Length of the data buffer.
((ENH POINTER))	A double word pointer to a buffer of enhancement codes to be used to enhance the remaining unshifted area.
((CHAR SET))	A double word pointer to a buffer of character set code characters to be used to change the character set state of the remaining unshifted area.
((ASCII POINTER))	A double word pointer to a buffer of ASCII data to be written into the remaining unshifted area.
(DIRECTION,DIST)	DIRECTION: 0 = up 1 = down 2 = left 3 = right DIST: The number of rows/columns to shift the current video data.

WRITE LINE.

This function writes a single row (or part of a row) in the workspace. Unlike Write Area, this intrinsic ignores the area bounds set by Define Area. If the position and length of the data are defined such that the right workspace boundary is violated, that portion of the data exceeding the boundary is ignored. No line wrap occurs.

(0, 7)	Function Code
(WKSP-ROW,WKSP-COL)	Defines the workspace relative position at which the data will be written.
(DATA LENGTH)	Length of the data buffers.
((ENH POINTER))	A double word pointer to the buffer of enhancement data to be written at the designated position.
((CHAR SET))	A double word pointer to a buffer of character set code characters to be used to change the character set state of the update area.
((ASCII POINTER))	A double word pointer to the buffer of ASCII data to be written at the designated position.

Application Softkeys

This section gives the AGIOS function calls that support Application Softkeys. Refer to Section 4 for general information about Application Softkeys. You can access these softkeys by typing [Shift] [User System]. You can use ASCII and Extended Roman characters within Application Softkey labels.

UPDATE SOFTKEY LABEL.

This function will update a softkey label and enhancement.

(0, 8)	Function Code
(,NUMBER)	Softkey Number (the softkey number is from 1 to 8 inclusive)
((DATA))	A double word pointer to the buffer of ASCII data to be written into the label area. (16 bytes.)
(,TOP ENH)	Enhancement code for the top half of the label.
(,BOT ENH)	Enhancement code for the bottom half of the label.

READ SOFTKEY LABEL.

This function gets the softkey number specified by the caller and then returns the softkey label and the enhancement code characters in two buffers.

(0, 9)	Function Code
(,NUMBER)	Softkey Number
((DATA))	A double word pointer to the buffer which is for the ASCII data.
((ENHANCEMENTS))	A double word pointer to the buffer which is for the enhancement code characters.

DISPLAY SOFTKEY LABELS.

This function displays the application softkey labels in the softkey window.

(0,11)	Function Code

Control Functions

EXECUTE TWO CHARACTER SEQUENCE (ESC CHAR).

(0,16)	Function Code
OP-CHAR	The character equivalent of a 2 character escape sequence. Any operation characters are valid except for those that return data. The following list defines some of the most common ones.

0	Dump Alpha to Printer
1	Set tab
2	Clear tab
3	Clear all tabs
4	Set left margin
5	Set right margin
9	Clear margins
@	Delay one second
A	Cursor up
B	Cursor down
C	Cursor right
D	Cursor left
E	Reset terminal
F	Home down
G	Return
H	Home up
I	Tab
J	Clear display
K	Clear line
L	Insert Line
L	Delete Line
P	Delete Character w/o Wrap
Q	Insert Character w/o Wrap
R	Insert Character Off
S	Roll Up
T	Roll Down
U	Next Page
V	Previous Page
W	Format Mode On
X	Format Mode Off
Y	Display Functions On
Z	Display Functions Off
[	Start Unprotected Field
]	End Protected Field
b	Enable Keyboard
c	Disable Keyboard
f	Modem Disconnect
g	Soft Reset Terminal
h	Home Up
i	Back Tab
j	Display Softkey Def Menu
k	Exit Softkey Def Menu
l	Memory Lock On
m	Memory Lock Off
p	Default [f1] Value
q	Default [f2] Value
r	Default [f3] Value
s	Default [f4] Value
t	Default [f5] Value
u	Default [f6] Value
v	Default [f7] Value
w	Default [f8] Value

POSITION CURSOR (ESC & a)

(0,17)	Function Code
MODE	Bit 0: 1 = window row address 0 = workspace row address Bit 1: 1 = relative row address 0 = absolute row address Bit 2: 1 = negative row address 0 = positive row address Bit 3: 1 = row address is valid 0 = row address is not valid Bit 4: 1 = window column address 0 = workspace column address Bit 5: 1 = relative column address 0 = absolute column address Bit 6: 1 = negative column address 0 = positive column address Bit 7: 1 = column address is valid 0 = column address is not valid
(COLUMN)	An unsigned integer
(ROW)	An unsigned integer

DEFINE ENHANCEMENTS (ESC & d).

(0,18)	Function Code
SEC	SEC: 1 = on, 0 = off.
ENH	ENH: the ESC &d code character (@..O)

CURSOR SENSE ABSOLUTE (ESC a).

(0,19)	Function Code
((BUFFER))	A pointer to a buffer where two words are returned. The first word is the column number in binary and the second word is the row number in binary.

CURSOR SENSE RELATIVE (ESC).

(0,20)	Function Code
((BUFFER))	A pointer to a buffer where two words are returned. The first word is the column number number in binary and the second word is the row number in binary.

SET CURSOR TYPE.

This function sets the alpha cursor type.

(0,21)	Function Code
(,TYPE)	Alpha cursor type: 0 = underscore 1 = inverse cell.

READ CURSOR TYPE.

This function reads the alpha cursor type.

(0,22)	Function Code
((BUFFER))	A pointer to a word location where alpha cursor type data is stored.

READ TERMINAL CONFIGURATION.

This function reads the current terminal configurations.

(0,24)	Function Code
((BUFFER))	A word pointer to the buffer where the current configuration is returned.

When this function is complete, BUFFER contains:

(,RRRRRTSP)	R = reserved bits, T = set if touch screen off, S = set if softkeys on, P = set if remote port 2.
(KEYBOARD LANGUAGE)	
(STRING LANGUAGE)	
(OP SYS DEVICE)	Bits 0-2: addr. 0-7. Bits 3-15: dev. 0 = HP-IB, 1 = Accsy.

Touch Screen Functions

The touch features on the HP 150 can be programmed in a variety of ways. The two general types of touch operations are "Field" operations and "Row/Column" operations. These two types can be intermixed.

Several of the touch screen function calls in this section assume keycode mode. Refer to "Keycode Modes" in Section 7 for information on this mode.

FIELD OPERATIONS.

There are four types of touch fields you can define. They are:

ASCII Fields:

This mode is very similar to the User-Definable Softkeys (see Section 4). A buffer of ASCII characters is associated with a touch field. A response string of 0 to 80 ASCII characters is obtained by consecutive keyboard input operations. The first input obtains the first ASCII byte, and the second input obtains the second ASCII byte, etc. The response string is generated when the field is touched and should be indistinguishable from the typing of the same string from the keyboard. Auto-repeat is performed.

Keycode Fields:

Keycode fields require that you be in keycode mode (see "Keycodes", Section 7). The two data words of the response string are treated as a keycode and a qualfier and are processed by the regular keyboard routines. The final response to touch depends on the state and mode of keyboard processing. Touch simulates typing on the keyboard. Releasing simulates releasing your finger from the key. Auto- repeat is performed.

Toggle Fields:

The touch field is defined as a toggle switch. Touching the area toggles the field on and off. Whenever the field is touched, sensing information is passed to the application. The sensing information consists of three data bytes. The data is obtained by three consecutive keyboard input operations. The qualifier word of each data byte returned to the application has the touch screen ID. The three data bytes of sensing information are:

```
01H - toggle on field report opcode
d1  - response string first byte
d2  - response string second byte

02H - toggle off field report opcode
d1  - response string first byte
d2  - response string second byte
```

Normal Fields:

This type of touch field senses touch and/or release. The sensing information consists of three data bytes. The data can be obtained by three consecutive keyboard input operations. The qualifier word of each data byte returned to the application has the touch screen ID. Auto-repeat is performed. The three data bytes of sensing information are:

```
05H - field touched report opcode
d1  - response string first byte
d2  - response string second byte

06H - field released report opcode
d1  - response string first byte
d2  - response string second byte
```

Touch fields can overlap. If they do, then the most recent definition for a character cell takes precedent.

ROW COLUMN OPERATIONS.

This type of touch operation returns the row and column position when a touch occurs. The row and column position are returned byte by by byte using the keyboard input function of the operating system. Three data bytes are returned. The qualifier word returned with each byte of data has the touch screen ID. The data bytes for row/column operations are:

```
03H - row column touch report opcode
row - touched row number in binary
col - touched column number in binary
```

The data bytes for release report of row/column are:

```
04H - row column release report opcode
row - touched row number in binary
col - touched column number in binary
```

DEFINE TOUCH FIELD (ESC - z g).

(0,32)	Function code
((STRING))	Pointer to response string. Points to 2 words for keycode field the first word is the qualifier and the second word is the keycode. Points to 2 bytes for toggle or normal field, 0 - 80 bytes for ASCII field.
(LENGTH)	Response string length
(ATTRIBUTE,MODE)	Touch ATTRIBUTE: 1= ASCII field 2= Keycode field 3= Toggle field 4= Normal field Reporting MODE: 1= Report when touched 2= Report when released 3= Report both touch and release
(ON-ENH,OFF-ENH)	Enhancements of the field for on and off state for toggle field. Also enhancements of the field when touched and released for normal, ASCII, and keycode fields.
(CURSOR,BEEP)	CURSOR: 0 = do not position cursor 1 = position cursor on touch BEEP: 0 = do not beep 1 = beep on touch
(LR-ROW,LR-COL)	Row and column of the lower right corner of the touch field.
(UL-ROW,UL-COL)	Row and column of the upper left corner of the touch field.

DEFINE SOFTKEY FIELD (ESC - z s).

This function defines one of the eight softkey label areas as a touch field. These fields when touched produce the same response as if the corresponding function key is typed. The default is all softkey touch fields are on.

(0,33)	Function code
(MODE,KEY)	KEY (Softkey number): 1-8 MODE: 1 = on, 0 = off.

DELETE TOUCH FIELD (ESC - z d).

Deletes the touch field with upper left corner at <row> <col>. Nothing happens if there is no touch field there. The row and column are screen relative coordinates.

(0,34)	Function code
(UL-ROW,UL-COL)	Row and column position of the field to be deleted. (0FFH,0FFH) deletes all fields.

TOUCH SCREEN RESET (ESC - z j).

Resets all fields to off.

(0,35)	Function code

SET TOUCH REPORTING MODES (ESC - z n).

This function determines if, and how, touch is to your application by the HP 150 terminal.

(0,36)	Function code
(,SCREEN-MODE)	Touch Field and Row/Col sensing:
0 -	Disable reporting.
1 -	Enable sensing for row/column position. Touch fields are inactive.
2 -	Enable sensing for touch fields only. Row/column sensing is inactive.
3 -	Enable sensing for both row/column and touch fields. Row/column sensing occurs for areas not defined as touch fields.
4 -	Toggles touch screen on and off.
10-14	- Same as 0-4, but causes escape sequence reports to be sent. This form is used ONLY by the system parser.
(,TOUCH-MODE)	Sense touch or touch-release: (used with Row/Col sensing only) 1 - Report on Touch 2 - Report on Release 3 - Report on both Touch and Release

Keyboard Intercept

Keyboard Intercept functions let you gain more control over the use of the keyboard. They are listed in the Keycode Table which follows. Each of the keyboard functions can be individually set to normal processing or one of the special processing modes. It should be noted that the keycodes for [f1] through [f12] are valid only when the application softkey labels are displayed on the screen with AGIOS function call (0,11).

There is no explicit "get keycode and qualifier" function call . The standard operating system console input function returns the normal ASCII code and also keycodes. The qualifiers are also returned if keycode mode is on.

The qualifier word is composed of the following bit values:

Bit	Value
15-8	Input Device ID: 0C0H = keyboard 080H = touch screen 000H = terminal internal
7	Special key. If set, the data is a non-ASCII keycode.
6	Reserved.
5	Left extend char - set when down.
4	Right extend char - set when down.
3	Control - set when down.
2	Left shift - set when down.
1	Right shift - set when down.
0	Repeating key when set.

To properly use the key intercept functions, the application program should first put the operating system's console input device into raw mode. This will allow keycodes to be passed through without interpretation by the operating system. Keycode mode should then be turned on. Finally each of the keys on the keyboard can be set to the desired mode of operation. For details on keycodes, see "Keycodes" in Section 7.

KEYCODE TABLE.

Key(s)	Intercept Code	Key(s)	Intercept Code
F1	00H	I-LINE	44H
F2	01H	D-LINE	45H
F3	02H	Shift/I-CHAR	46H
F4	03H	Shift/D-CHAR	47H
F6	05H	Shift/C-DISP	48H
F7	06H	Shift/C-LINE	49H
F8	07H	ENTER	50H
F9	08H	SYSTEM	51H
F10	09H	SELECT	52H
F11	0AH		
F12	0BH	MENU	53H
		BREAK	54H
C-UP	20H	ESC	55H
C-DOWN	21H	CAPS	56H
C-RIGHT	22H	DEL	57H
C-LEFT	23H	STOP	58H
TAB	24H	Shift/MENU	59H
RET	25H	Shift/STOP	5AH
B-TAB	26H	Shift/ENTER	5BH
BSPACE	27H	Cntrl/ENTER	5CH
R-DOWN	28H		
R-UP	29H	G-C-LEFT	61H
R-LEFT	2AH	G-C-DOWN	62H
R-RIGHT	2BH	G-C-RIGHT	63H
H-UP	2CH	G-R-UP	64H
H-DOWN	2DH	G-C-UP	65H
N-PAGE	2EH	G-R-LEFT	66H
P-PAGE	2FH	G-R-DOWN	67H
		G-R-RIGHT	68H
N-ZERO	30H	G-CLEAR	69H
N-ONE	31H	G-PAD-TOGGLE	6AH
N-TWO	32H	A-DSPLY	6BH
N-THREE	33H	G-CURSOR	6CH
N-FOUR	34H	G-DSPLY	6DH
N-FIVE	35H	G-H-UP	6EH
N-SIX	36H	G-H-DOWN	6FH
N-SEVEN	37H		
N-EIGHT	38H	N-MINUS	70H
N-NINE	39H	N-ASTERISK	71H
		N-PLUS	72H
I-CHAR	40H	N-SLASH	73H
D-CHAR	41H	N-COMMA	74H
C-DISP	42H	N-ENTER	75H
C-LINE	43H	N-TAB	76H
		N-PERIOD	77H

DEFINE KEY CHARACTERISTICS.

This function lets you alter characteristics of any of the special keys. Specifically, you can:

- Process the key normally (Same as on HP 2623 terminal)
- Intercept the key and pass a keycode to the application for processing
- Ignore the key when it is pressed
- Beep when the key is pressed in combination with the above characteristics

(0,40)	Function Code
(CHARACTERISTICS)	Key Characteristics Bit: Action: 0 beep 1 intercept 2 ignore 3-15 reserved If bit 1 and 2 are both set, the key is treated as an intercept key. When both are zero, the key resumes normal functioning.
(KEYCODE)	Keycode from table on previous page. A value of 0FEH will set all special keys to the specified characteristics.

GET KEY CHARACTERISTICS.

This function returns the characteristics of a key to the caller.

(0,41)	Function Code
((BUFFER))	Pointer to a buffer where the key's characteristics will be returned.
(KEYCODE)	Keycode

PUT KEY.

This function lets you specify direct the terminal to process the keycode normally. Use this function when you wish to 'process' a keypress which was read in when intercept mode was active. For normal ASCII keys, you would simply 'echo' the character in place of using this function.

(0,42)	Function Code
(QUALIFIER)	Qualifier

Bit: Interpretation:

15-8	= Input Device ID (0C0H = keyboard)
7	= Special key. Must be set.
6	= Reserved
5	= Left extend char, set when down
4	= Right extend char, set when down
3	= Control, set when down
2	= Left shift, set when down
1	= Right shift, set when down
0	= Not used

(KEYCODE)	Keycode

KEYCODE ON/OFF.

This function turns the keycode mode of the console device on and off. If keycode mode is off, each key hit on the keyboard returns one byte of data when the console input device is read. If keycode mode is on, each key press returns four bytes of data. The first two bytes form a word of qualifiers and the next two bytes form a word of key data. See 'Keycode Modes' in Section 7 for more detail.

(0,43)	Function Code
MODE	Keycode mode:

1 = on
0 = off

KEYCODE STATUS.

This function returns the on/off status of a keycode.

(0,44)	Function Code
((BUFFER))	A pointer to a byte location where the keycode on/off status is returned.

READ KEYPAD STATUS.

This function returns the status of whether the extended keypad is in numeric or graphics mode.

(0,44)	Function Code
((BUFFER))	A pointer to a byte location where the keypad status is returned.

When this function is complete, BUFFER contains 0 if numeric mode is set, and 1 if graphics mode is set.

Display Control (ESC * d)

CLEAR GRAPHICS MEMORY (ESC * d a).

This function clears graphics display memory to 0. The complete displayable graphics area of 512 x 390 dots are cleared.

(4, 1) Function Code

SET GRAPHICS MEMORY (ESC * d b).

This function sets graphics display memory to 1. The complete displayable graphics area of 512 x 390 dots are set.

(4, 2) Function Code

TURN ON GRAPHICS DISPLAY (ESC * d c).

This function turns on the graphics display. The data in graphics memory is not affected.

(4, 3) Function Code

TURN OFF GRAPHICS DISPLAY (ESC * d d).

This function turns off the graphics display. The data in graphics memory is not affected.

(4, 4) Function Code

TURN ON ALPHANUMERIC DISPLAY (ESC * d e).

This function turns on the alphanumeric display and alphanumeric cursor. The data in alphanumeric memory is not affected.

(4, 5)	Function Code

TURN OFF ALPHANUMERIC DISPLAY (ESC * d f).

This function turns off the alphanumeric display. The data in alphanumeric memory is not affected.

(4, 6)	Function Code

TURN ON GRAPHICS CURSOR (ESC * d k).

This function turns on the graphics cursor. The data in graphics memory is not affected.

(4, 7)	Function Code

TURN OFF GRAPHICS CURSOR (ESC * d l).

This function turns off the graphics cursor. The data in graphics memory is not affected.

(4, 8)	Function Code

TURN ON RUBBER BAND LINE (ESC * d m).

This function turns on the rubber band line and graphics cursor.

(4, 9)	Function Code

TURN OFF RUBBER BAND LINE (ESC * d n).

This function turns off the rubber band line.

(4,10)	Function Code

MOVE GRAPHICS CURSOR ABSOLUTE (ESC * d <x,y> o).

This function moves the graphics cursor to the specified location. The move occurs even if the cursor is turned off.

(4,11)	Function Code
(X-COORD)	The X coordinate of the new cursor position expressed as an absolute number in the range of plus and minus 16383.
(Y-COORD)	The Y coordinate of the new cursor position expressed as an absolute number in the range of plus and minus 16383.

MOVE GRAPHICS CURSOR INCREMENTAL (ESC * d <x,y> p).

This function moves the graphics cursor to the specified location. The move occurs even if the cursor is turned off.

(4,12)	Function Code
(X-COORD)	The X coordinates of the new cursor position expressed as a number that is relative to the current cursor position. Its range extends from -32768 to +32767.
(Y-COORD)	The Y coordinate of the new cursor position expressed as a number that is relative to the current cursor position. Its range extends from -32768 to +32767.

TURN ON ALPHANUMERIC CURSOR (ESC * d q).

This function turns on the alphanumeric cursor. The data in alphanumeric memory is not affected.

(4,13)	Function Code

TURN OFF ALPHANUMERIC CURSOR (ESC * d r).

This function turns off the alphanumeric cursor. The data in alphanumeric memory is not affected.

(4,14)	Function Code

TURN ON GRAPHICS TEXT MODE (ESC * d s).

This function turns on graphics text mode. Characters that normally go to the alphanumeric display will be drawn on the graphics display.

(4,15)	Function Code

TURN OFF GRAPHICS TEXT MODE (ESC * d t).

This function turns off graphics text mode.

(4,16)	Function Code

Vector Drawing Mode (ESC * m)

SELECT DRAWING MODE (ESC * m <mode> a).

This function selects the vector drawing mode.

(4,17)	Function Code
(MODE)	Drawing Mode:

0 = Graphics memory not changed
1 = Clear mode
2 = Set mode
3 = Complement mode
4 = Jam mode

SELECT LINE TYPE (ESC * m <type> b).

This function selects the vector line type.

(4,18)	Function Code
(TYPE)	Line Type:

1 = ————————————————————
2 = User defined line pattern
3 = Current area fill pattern
4 = —— – —— – —— – —— – —— – —— – —— –
5 = ———— ———— ———— ———— ————
6 = —— —— —— —— —— —— —— ——
7 = ...
8 = ——·——·——·——·——·——·——·——·——·
9 = --- --- --- --- --- --- --- --- --- ---
10= —— – – —— – – —— – – —— – – —
11= Point plot

DEFINE LINE PATTERN AND SCALE (ESC * m <pattern><scale> c).

This function defines a user line pattern and scale.

(4,19)	Function Code
(,PATTERN)	The line pattern expressed as an eight bit binary number.
(SCALE)	The line scale expressed as a binary number from 1 to 16.

DEFINE AREA FILL PATTERN (ESC * m <pattern> d).

This function defines a user area fill pattern of 8 by 8 screen dots.

(4,20)	Function Code
(,DATA ROW1) (,DATA ROW2) (,DATA ROW3) (,DATA ROW2) (,DATA ROW4) (,DATA ROW5) (,DATA ROW6) (,DATA ROW7) (,DATA ROW8)	You specify all eight rows of the 8 by 8 fill pattern. Each DATA-ROW is an eight-bit byte which defines a particular row of the pattern.

FILL RECTANGULAR AREA, ABSOLUTE (ESC * m <x1,y1,x2,y2> e).

This function fills a rectangular area with the selected line or area fill pattern. The rectangular region is defined by specifying the lower left and upper right coordinates.

(4,21)	Function Code
(LWR LEFT X-COORD) (LWR LEFT Y-COORD) (UPR RIGHT X-COORD) (UPR RIGHT Y-COORD)	Each coordinate is in the range of -16384 to +16383.

FILL RECTANGULAR AREA, RELOCATABLE (ESC * m <x1,y1,x2,y2> f).

This function fills a rectangular area with the selected line or area fill pattern. The rectangular region is defined by specifying the lower left and upper right coordinates.

(4,22)	Function Code
(LWR LEFT X-COORD) (LWR LEFT Y-COORD) (UPR RIGHT X-COORD) (UPR RIGHT Y-COORD)	Each value is in the range from -32768 to +32767.

SELECT POLYGONAL FILL PATTERN (ESC * m <pattern> g).

This function selects a pattern for polygonal and rectangular area fill.

(4,23)	Function Code
(PATTERN)	Area Fill Pattern: 1 = Solid fill pattern 2 = User-defined fill pattern 3-10 = Pre-defined fill pattern

SELECT BOUNDARY PEN (ESC * m <pen> h).

This function selects the pen to be used to draw the boundary of a filled polygon. The actual value is not significant in a black and white system. This function turns on boundary drawing with a solid line pattern. The drawing of each edge of the boundary can be individually controlled.

(4,24)	Function Code
(PEN)	Boundary Pen Number

NO POLYGON BOUNDARY (ESC * m h).

This function turns off drawing of boundary around a polygon.

(4,25)	Function Code

SET RELOCATABLE ORIGIN (ESC * m <x,y> j).

This function sets the relocatable origin to the specified absolute location.

(4,26)	Function Code
(X-COORD)	The X coordinate is the new relocatable origin expressed as an absolute number in the range of -16384 to +16383.
(Y-COORD)	The Y coordinate is the new relocatable origin expressed as an absolute number in the range of -16384 to +16383.

SET RELOCATABLE ORIGIN TO PEN POSITION (ESC * m k).

This function sets the relocatable origin to the current pen position.

(4,27)	Function Code

SET RELOCATABLE ORIGIN TO CURSOR POSITION (ESC * m l).

This function sets the relocatable origin to the current cursor position.

(4,28)	Function Code

Graphics Text (ESC *)

The HP 150 offers a comprehensive graphics character set in read-only memory (ROM). This standard character set is used by all graphics text operations. However, you do have the ability to create custom characters of your own design and to use these singly or to replace the entire built-in character set.

SET GRAPHICS TEXT SIZE (ESC * m <size> m).

This function sets the graphics text size. The vector lists that define the current character set are scaled using this text size.

(4,29)	Function Code
(X-SCALE)	The X coordinate scale factor for text characters. The format is a 16 bit number with the radix point between bits 7 and 8: Bits 1-8 = integer Bits 9-16 = fraction
(Y-SCALE)	The Y coordinate scale factor for text characters. The format is a 16 bit number with the radix point between bits 7 and 8: Bits 1-8 = integer Bits 9-16 = fraction

SET GRAPHICS TEXT ORIENTATION (ESC * m <orientation> n).

This function selects the graphics text orientation. This also changes the direction of line feed, carriage return, and backspace. The desired orientation is specified by a number defined as:

(4,30)	Function Code
(ORIENTATION)	Graphics Text Orientation: 1 = Normal 2 = Rotate 90 degrees counterclockwise 3 = Rotate 180 degrees counterclockwise 4 = Rotate 270 degrees counterclockwise

TURN ON TEXT SLANT (ESC * m o).

This function turns on the 26.57 degree slant of graphics text characters.

(4,31)	Function Code

TURN OFF TEXT SLANT (ESC * m p).

This function turns off 26.57 degrees slant of graphics text characters.

(4,32)	Function Code

SET GRAPHICS TEXT ORIGIN (ESC * m <0-9> q).

This function sets the graphics text origin to one of twelve positions of text justification. The positions are shown in this figure:

```
                3------6------9
                |             |
                |             |
                |             |
                |             |
                2      5      8
                |             |
  Base line ->  0     10     11
                |             |
                |             |
                1------4------7
```

(4,33)	Function Code
(ORIGIN)	Graphics Text Origin:
	A number from 0 to 11.

GRAPHICS TEXT LABEL (ESC * l <text>).

This function outputs a string of graphics characters.

(4,34)	Function Code
((TEXT))	Segment and offset address of a string of characters. The string must be terminated by CR, LF, CR LF, or LF CR.

DEFINE USER CHARACTER SET.

This function lets you re-define the entire graphics character set. All subsequent graphics text operations will use this character set. This includes text size, orientation, slant, and justification.

(4,35)	Function Code
((TABLE))	Segment and offset address of the table that points to the vector lists of characters.

SELECT DEFAULT CHARACTER SET.

This function sets the character set to the default set maintained by the system. The cell size is 7 x 10.

(4,36)	Function Code

OUTPUT SINGLE TEXT CHARACTER.

This function outputs a single graphics character defined by a vector list. All current graphics text operations such as size and orientation apply. See 'Graphics Text' in Section 7 for detailed information.

(4,37)	Function Code
((CHARACTER))	Segment and offset address of the vector list of a single character.

SET GRAPHICS DEFAULT (ESC * m r).

This function sets the graphics parameters to their default values.

(4,38)	Function Code

The defaults affected by this call are:

Pen down
Line type 1
User-defined line pattern solid
User-defined area fill pattern solid
Boundary pen off
Drawing mode set
Relocatable origin 0,0
Text size 1
Text origin 1
Text slant off
Text orientation 1
Graphics text off
Graphics display on
Alphanumeric display on
Graphics cursor off
Alphanumeric cursor on
Rubber band line off
Graphics cursor position 0,0

SET PICTURE DEFINITION DEFAULTS (ESC * m 1 r).

This function sets the picture definition parameters to their default values.

(4,72)	Function Code
(RESET LEVEL)	The level of graphics reset. On the HP 150 the value '1' is the only supported level.

The picture defaults are:

Pen down
Line type 1
User-defined line pattern solid
User-defined area fill pattern solid
Boundary pen off
Drawing mode set
Text size 1
Text origin 1
Text slant off
Text orientation 1
Graphics text off

GRAPHICS HARD RESET (ESC * w r).

Sets the graphics parameters to their power on state.

(4,73)	Function Code

Graphics Plotting (ESC * p)

LIFT PEN (ESC * p a).

This function lifts the pen.

(4,39)	Function Code

VECTOR MOVE (ESC * p a <x,y>).

This function lifts the pen and moves the pen to the new coordinate position. The pen is lowered at the end of the operation.

There are three ways to specify the new coordinate position:

(4,40)	Function Code
(X-COORD) (Y-COORD)	The X and Y numbers give an absolute coordinate position in the range from -16384 to +16383.
(4,41)	Function Code
(X-COORD) (Y-COORD)	The X and Y numbers give an incremental coordinate position in the range from -32768 to +32767.
(4,42)	Function Code
(X-coord) (Y-coord)	The X and Y numbers give a relocatable coordinate position in the range from -32768 to +32767.

LOWER PEN (ESC * p b).

This function lowers the pen.

(4,43)	Function Code

VECTOR DRAW (ESC * p b <x,y>).

This function lowers the pen and draws a vector to the new coordinate position. The pen is lowered at the end of the operation.

There are three ways to specify the new vector coordinates:

(4,44)	Function Code
(X-COORD) (Y-COORD)	The X and Y numbers give the absolute coordinates of the vector position. They are in the range from -16384 to +16383.
(4,45)	Function Code
(X-COORD) (Y-COORD)	The X and Y numbers give the incremental coordinates of the vector position. They are in the range from -32768 to +32767.
(4,46)	Function Code
(X-COORD) (Y-COORD)	The X and Y numbers give the relocatable coordinates of the vector position. They are in the range from -32768 to +32767.

PLOT TO CURSOR POSITION (ESC * p c).

This function moves the pen from its current position to the current cursor position if the pen is up. A draw is performed from the current pen position to the current cursor position if the pen is down.

(4,47)	Function Code

POINT PLOT (ESC * p d).

This function draws a dot at the current pen position and then lifts the pen.

(4,48)	Function Code

SET RELOCATABLE ORIGIN TO PEN POSITION (ESC * p e).

This function sets the relocatable origin to the current pen position.

(4,49)	Function Code

START POLYGONAL AREA FILL (ESC * p s).

This function starts polygonal area fill. The boundary pen is lowered with this function.

(4,50)	Function Code

TERMINATE POLYGONAL AREA FILL (ESC * p t).

This function terminates the polygon definition and fills the polygon.

(4,51)	Function Code

POLYGON MOVE.

This function closes the polygon defined up to this point and moves the pen to the new coordinate position to start a new polygon.

There are three ways to specify the new coordinate position:

(4,52)	Function Code
(X-COORD) (Y-COORD)	The X and Y numbers give the absolute coordinates of the new position. They are in the range from -16384 to +16383.
(4,53)	Function Code
(X-COORD) (Y-COORD)	The X and Y numbers give the incremental coordinates of the new position. They are in the range from -32767 to +32767.
(4,54)	Function Code
(X-COORD) (Y-COORD)	The X and Y numbers give the relocatable coordinates of the new position. They are in the range from -32768 to +32767.

POLYGON DRAW.

This function defines the edge of a polygon from the current pen position to the new coordinate position.

There are three ways that you can specify the new coordinate position:

(4,55)	Function Code
(X-COORD) (Y-COORD)	The X and Y numbers give the absolute coordinates of the new position. They are in the range from -16384 to +16383.
(4,56)	Function Code
(X-COORD) (Y-COORD)	The X and Y numbers give the incremental coordinates of the new position. They are in the range from -32768 to +32767.
(4,57)	Function Code
(X-COORD) (Y-COORD)	The X and Y numbers give the relocatable coordinates of the new position. They are the range from -32768 to +32767.

LIFT BOUNDARY PEN (ESC * p u).

This function lifts the polygon boundary pen. Undrawn edges of the polygon are not drawn. This remains in effect until the boundary pen is lowered.

(4,58) Function Code

LOWER BOUNDARY PEN (ESC * p v).

This function lowers the polygon boundary pen. If a boundary pen has been specified, undrawn edges of the polygon are drawn with a solid line pattern. This remains in effect until the boundary pen is lifted.

(4,59) Function Code

Graphics Status (ESC * s)

READ DEVICE ID (ESC * s 1 ∧).

This function returns the device id of the HP 150.

(4,60)	Function Code
((BUFFER))	Segment and offset address of the buffer to be used for returned device data.

When this function is complete, BUFFER contains an ASCII string that identifies the device.

READ PEN POSITION (ESC * s 2 ∧).

This function returns the current position and the state of the pen.

(4,61)	Function Code
((BUFFER))	Segment and offset address of the buffer to be used for returned pen status data.

When this function is complete, BUFFER contains:

(X-COORD) (Y-COORD)	The binary X and Y coordinates of the current pen position
(STATE)	0 = pen lifted, 1 = pen lowered

READ CURSOR POSITION (ESC * s 3 ^).

This function returns the current position of the cursor.

(4,62)	Function Code
((BUFFER))	Segment and offset address of the buffer to be used for the returned cursor data.

When this function is complete, BUFFER contains:

(X-COORD) (Y-COORD)	The X and Y coordinates of current cursor position.

READ CURSOR POSITION, WAIT FOR KEY (ESC * s 4 ^).

This function returns the current position of the cursor, but lets the user move it on the display first. The user can type any ASCII key or the SELECT key on the keyboard to move the cursor. As soon as one of these characters is typed, the cursor coordinates are returned to the program.

(4,63)	Function Code
((BUFFER))	Segment and offset address of the buffer to be used for returned cursor position.

When this function is complete, BUFFER contains:

(X-COORD) (Y-COORD)	The X and Y coordinates of current cursor position.
(CODE)	The character code of the key that was typed.

READ DISPLAY SIZE (ESC * s 5 ^).

This function returns the number of displayable units and also the number of units in millimeters.

(4,64)	Function Code
((BUFFER))	Segment and offset address of the buffer to be used for the returned displayable size and unit data.

When this function is complete, BUFFER contains:

(X-LWR-LEFT) (Y-LWR-LEFT) (X-UPR-RIGHT) (Y-UPR-RIGHT)	The X and Y coordinates of the maximum display size.
(X-MM) (Y-MM)	The X and Y dimensions in dots / millimeters.

READ GRAPHICS SETTINGS (ESC * s 6 ^).

This function returns information about the current graphics settings in effect.

(4,65)	Function Code
((BUFFER))	Segment and offset address of the buffer to be used for returned graphics settings.

When this function is complete, BUFFER contains the settings in consecutive words:

(CLEAR DISPLAY)
(NUMBER OF PENS)
(RESERVED)
(RESERVED)
(AREA SHADING)
(RESERVED)
(RESERVED)
(DYNAMIC MODIFICATION)
(GRAPHICS CHARACTER SIZE)
(GRAPHICS CHARACTER ANGLES)
(GRAPHICS CHARACTER SLANT)
(DOT-DASH LINE PATTERN)
(RESERVED)
(RESERVED)
(RESERVED)
(RESERVED)

READ GRAPHICS TEXT STATUS (ESC * s 7 ^).

This function returns the current attributes of graphics text.

(4,66)	Function Code
((BUFFER))	Segment and offset address of the buffer to be used for the returned graphics attributes.

When this function is complete, BUFFER contains:

(X SIZE) (Y SIZE)	The character cell size.
(ORIGIN)	The text origin.
(ANGLE)	The text orientation.
(SLANT)	The character slant.

READ ZOOM STATUS (ESC * s 8 ^).

This function returns the terminal's zoom setting.

(4,67)	Function Code
((BUFFER))	Segment and offset address of the buffer to be used for the returned zoom setting.

When this function is complete, BUFFER contains:

(ZOOM SIZE)	1 - 16 (the HP 150 always returns 1).
(ZOOM ON/OFF)	0 = off, 1 = on (the HP 150 always returns 0).

READ RELOCATABLE ORIGIN (ESC * s 9 ^).

This function returns the current relocatable origin.

(4,68)	Function Code
((BUFFER))	Segment and offset address of the buffer to be used for the returned origin.

When this function is complete, BUFFER contains:

(X-COORD) (Y-COORD)	The X and Y coordinates of the current relocatable origin.

READ RESET STATUS (ESC * s 10 ^).

This function returns information on whether the terminal has executed a full reset since the last time reset status was checked.

(4,69)	Function Code
((BUFFER))	Segment and offset address of the buffer to be used for the returned reset status.

When this function is complete, BUFFER contains:

(RESET STATUS)
(RESERVED)
(RESERVED)
(RESERVED)
(RESERVED)
(RESERVED)
(RESERVED)
(RESERVED)

READ AREA SHADING (ESC * s 11 ∧).

This function returns information on the area shading capability of the terminal.

(4,70)	Function Code
((BUFFER))	Segment and offset address of the buffer to be used for the returned shading data.

When this function is complete, BUFFER contains:

(CAPABILITIES)	The area shading capabilities.
(WIDTH) (HEIGHT)	The area shading pattern size.

READ DYNAMICS (ESC * s 12 ∧).

This function returns information on the terminal's dynamic graphics capabilities.

(4,71)	Function Code
((BUFFER))	Segment and offset address of the buffer to be used for the returned dynamic graphics data.

When this function is complete, BUFFER contains:

(SELECTIVE-ERASE-CAPABILITIES)

(COMPLEMENT-CAPABILITIES)

READ EXTENDED SCREEN DIMENSIONS.

This function provides information about the alpha and graphics screen size plus the relationship between the two.

(0,74)	Function Code
((BUFFER))	Segment and offset address of the buffer to be used for the returned screen size data.

When this function is complete, BUFFER contains:

(X-PIXELS)	512 graphics display size in pixels.
(Y-PIXELS)	390
(ROWS)	27 alpha display size.
(COLUMNS)	80
(X-MM)	160 graphics display size in mm.
(Y-MM)	120
(ROW-MM)	150 alpha display size in mm.
(COL-MM)	116
(DELTA-X)	10 graphics origin minus alpha origin in mm.
(DELTA-Y)	4

THIS PAGE SHOULD BE BLANK

USING P.A.M. AND FILE MANAGER

SECTION 6

WHAT IS P.A.M.?

P.A.M. (Personal Applications Manager) is a shell program around MS-DOS which offers a friendly interface for the user. For this reason, many application programs run under the control of P.A.M.

P.A.M. and a system program, File Manager (see "What is File Manager?" in this section), work together on your HP 150. Since they need many of the same routines to accomplish their tasks, they share as many routines and data areas as possible. Some of the shared routines are:

- User interface code (screen handler)
- Message file handler
- MS-DOS interface
- AGIOS interface

When you need more details on P.A.M., refer to the *HP 150 Personal Computer Owner's Manual.*

RUNNING YOUR PROGRAM UNDER P.A.M.

If you want the user of your application to run it from P.A.M., you need to create a control file on your master distribution disc. This file is used by the INSTALL program when the user of your application installs it onto a working disc. The INSTALL utility, in addition to installing your application onto a work disc, creates/updates an invisible file called, "PAM.VOL". It is this file that is searched for installed applications when a user touches Reread Discs.

NOTE

Because PAM.VOL is an invisible file, you cannot use the MS-DOS COPY utility to copy it to other discs. However, you can use TYPE to copy it when you redirect the output. This is a bug in MS-DOS. Future revisions of MS-DOS may fix this bug, so don't count on it being available indefinitely.

The control file that you create for INSTALL contains your application name, size requirements, and the names of the programs comprising your application. You can use most word processors, line editors or even the COPY command to create the control file on your master disc. The control file is a standard ASCII file. You give it your application name and append the file extension, ".INS". For example, if your application is a word processing program, you might name the control file, "WORD.INS".

The .INS File Format

The following table gives you the structure of the .INS file.

Line	Content	Comments
1	Label	The Application Name as displayed by P.A.M. on the main menu : 13 bytes maximum
2	Version	The application revision level : 7 bytes maximum
3	Main Prog	The full name of the program to be loaded by P.A.M. when this application is selected : 13 bytes maximum
4	Com Line	The command line to be passed to the Main Prog when execution starts : 67 bytes maximum
5.	Disc Sp	The amount of disc space required on the work disc before installation will proceed : 8 bytes maximum
6	Bl Line	A blank line (carriage return only)
7/n	Files	The list of file(s) to be copied to the work disc, one file per line.

A Sample .INS File

The following items show what a sample .INS file might look like for an accounting application from Kehoe Industries. To execute the application from COMMAND.COM, the command line is 'KI MAIN'. The .INS file shows:

```
KI Accounting
A.02.00
KI.EXE
MAIN
100000

KI.EXE
KIOVR1.OVR
KIOVR2.OVR
KIMSG.MSG
```

Installing From More Than One Disc

When your application extends to more that one disc, place an asterisk (*) in the .INS file at the point at which you want a new master disc to be inserted. The asterisk is placed on a line by itself anywhere from line 7 to the end of the .INS file. When INSTALL encounters the asterisk, it pauses and prompts the user to insert a new disc.

A Caution on Application Size

Remember when calculating the required disc space for your application that the sum of file sizes may not be adequate. When MS-DOS allocates disc space, it does so in clusters of 1K on 3.5 inch media and 4K on Winchester discs. To accurately provide INSTALL with the required disc space, you should reserve an even multiple of the media cluster size for each file. This may mean you need more than the actual number of bytes for your application. If all else fails, be sure the amount of disc you've specified is sufficient.

How P.A.M. Runs Your Program

P.A.M. runs your program by using the MS-DOS EXEC function, AX set to 4B00 hex which specifies the load and execute option.

Your application can be run only from the root directory. This occurs because the directory path name in PAM.VOL is not long enough to store the full 64 characters that may be required.

WHAT IS FILE MANAGER?

File Manager is the file utility portion of P.A.M. that is resident whenever PAM has been loaded. This subprogram provides functions such as directory searches, file copying and printing (many of the utilities available in COMMAND.COM). You can call File Manager the same way P.A.M. calls it (see the previous paragraph), or you can call it specifying your own paramters such as function key labels and banner line title. Specifying your own parameters makes File Manager appear to be an integral part of your application. The following paragraphs in this section explain how you call File Manager specifying these optional parameters.

For detailed information on File Manager, refer to the *HP 150 Personal Computer Owner's Guide*.

How File Manager is Invoked By P.A.M.

When P.A.M. calls File Manager, it clears the function key labels, takes the HP 150 out of keycode mode, etc. When File Manager is finished, and control is returned to P.A.M., shared buffers are reinitialized, all discs are read as indicated by PAM.VOL, and the installed applications are displayed.

USING FILE MANAGER FROM YOUR APPLICATION PROGRAM

This section tells you how to call the facilities of File Manager from your application program, and at the same time, specify some initial values to File Manager.

Issuing the Call to File Manager

To call File Manager, you need to write an assembly language routine that can be invoked from your application program. The first thing the assembly language routine does is to obtain the entry point address in memory of File Manager.

The entry point to File Manager is passed in the `program environment` available to your application from MS-DOS. The environment address is located at address 5CH in base page.

When P.A.M. executes your application, it passes the address of File Manager in the environment.

In the environment, your application will find the string "FILE_MANAGER=xxxxyyyy". Here "xxxx" is the segment and "yyyy" the offset within segment of the File Manager entry point. The string is followed by a null byte. By converting those ASCII characters to an effective address, you can call File Manager. There is a sample program on the ISV Development Disc which performs such a task. You can use it to model your own solution.

You will build a command buffer for File Manager (see below) and push its 32 bit address on the stack. Now you are ready to call File Manager. Do so with a 'long call'. When control returns to your application, the 32 bit address will have been removed (with a RET 4 instruction). The AX register will return as zero if File Manager was accessed successfully. A value of four (4) indicates an unsuccessful call. One field in the command is *Status*, and you can obtain additional information there. See the comments presented later with each of the four File Manager function requests.

Setting Up the Call Buffer

When you issue a File Manager call, parameters are passed to File Manager in a buffer that you set up. Its format depends on the operation (function) that you want to perform. You can specify these operations on a File Manager call:

- Simulate a P.A.M. call
- Get the Primary Printer String
- Specify the File Manager banner line and function key labels
- Perform automatic file lookups
- Specifying the File Manager displayed directory

The following Pascal record structure shows you how to set up the first two fields in this buffer. These two fields are always the same, regardless of the operation you're performing. The other fields are operation specific and are explained in the appropriate sections which follow.

If you're working in a language other that Pascal, use the word/character format and set your buffer up accordingly.

```
type fmrec = record
     func       :    integer  (one word)
     status     :    bitmap   (one word)
        .
        .
        .
        .
end
```

Func contains the operation you want to perform. The operations you can perform are explained in the following sections. *Status* tells whether the call parameters are valid and whether defaults are used.

More About the Status Field

The second entry in the record structure is a *status* word. When File Manager returns to your application program, the *status* word is set as follows:

Bit 0 =	File Manager was called, and exited normally (always set).
Bit 1 =	The File Manager parameters are invalid; the defaults are used.
Bit 2 =	The File Manager parameters are invalid; and there are no defaults. File Manager exited without doing anything (e.g. invalid function number).
Bits 3-15 =	These bit values depend on the function you requested. Refer to the specific function operations for information on these.

The *status* bits are not consecutive bits; that is, bit 0 is not the first bit in the word. The order is the typical 808x order, low byte first, then high byte. The following Pascal code suggest how you could access them and gives you their relative position in the word.

```
  type bits = (bit7, bit6, bit5, bit4, bit3, bit2,
               bit 1, bit 0, bit 15, bit 14, bit 13,
               bit12, bit11, bit10, bit9, bit8);

     bitmap = set of bits;
```

Simulating a P.A.M. Call (Function 0)

This function calls File Manager such that the displays appear exactly as if File Manager were called from P.A.M.

Call Buffer:

```
type   fmrec = record
       func     :    0            (One word)
       status   :    Returned     (One word)
end
```

Getting the Primary Printer String (Function 1)

This function returns the name of the primary printer that the user selected in File Manager. If the user did not select a primary printer, the returned string is "PRN:". Since this function merely retrieves information from File Manager, your application display remains unaltered.

Call Buffer:

```
type   fmrec = record
       func     :    1          (One word)
       status   :    Returned   (One word)

                     Bit 3 = The printer field has valid
                             information.

       printer :     Returned string  (66 bytes)
end
```

Note: You change the current primary printer by entering File Manager and selecting, Print then Set Printer.

Performing File Operations (Function 2)

This function lets you initialize File Manager to display the softkey labels and directories you choose, lets you enter File Manager to perform any disc operation normally available in File Manager, then lets you obtain information about the current discs when File Manager is finished.

For example, lets suppose you have a graphics application and you want to let the user do a directory lookup to find the file he wants to load. You may want to specify a particular directory to display initially in File Manager along with the appropriate application softkey labels. The user begins searching the directory you specified, and can continue to search others if necessary. After the file is found and the user exits from File Manager, you can retrieve the file name that was selected.

All of the file operations use the same parameter format which follows. When you want File Manager to use the default value for any of these fields, put a null character in the first position of the field.

Call Buffer:

```
type  fmrec = record
      func    :    2        (One word)
      status  :    Returned (One word)

                   Bit 3: rfile contains a file name.

                   Bit 4: rdir contains a directory name.

      title   :    Banner   (80 bytes)
      appl    :    Function key label, packed array (8 bytes)
      type    :    Function key label, packed array (8 bytes)
      ddir    :    Directory to display, packed array (66 bytes)
      rfile   :    Returned file name (66 bytes)
      rdir    :    Returned directory (66 bytes)
end
```

SPECIFYING THE BANNER LINE TITLE THAT FILE MANAGER USES.

Title lets you specify the File Manager banner. You set *title* equal to the full 80 characters you want to display.

SPECIFYING THE FUNCTION KEY LABELS THAT FILE MANAGER USES.

Appl and *type* are fields which contain the text to be used in two of the function key labels in File Manager.

Appl contains the text for "Applic" in Back To Applic. If you do not specify text, the default is "Applic".

Type contains the text for "File" in Applic File. If you do not specify text, the default is "File".

For example, if the GRAPHICS application uses the *appl* field containing the text, "Graphics", and a *type* field containing "Chart", File Manager would show the keys as Applic Chart and Back To Graphics.

SPECIFYING THE INITIAL DIRECTORY THAT FILE MANAGER DISPLAYS.

Ddir specifies the first directory screen that you see in File Manager. You specify the directory by entering the drive name (starting at the root) followed by either a wildcard name and a null, or a back slash (\) and a null.

If you do not use *ddir*, the last directory displayed is used.

GETTING A FULL FILE IDENTIFICATION FROM FILE MANAGER.

Rfile is the field where File Manager places the full file identification for any file that the user requests. File Manager returns the root drive, directory, path, file name and extension. If you do not want to have a full file identification returned, set the first character of *file* to null.

GETTING A DIRECTORY FROM FILE MANAGER.

Rdir is the field where File Manager places the directory path name. If you do not want to have a directory path name returned, set the first character in the *rdir* array to null.

Application Select Only (Function 3)

This function is identical to the File Operations (Function 2) except that the user cannot perform file operations such as copy and delete in File Manager (File Functions is not available). The user of your application can only select files and directories in File Manager.

When you want File Manager to use the default value for any of these fields, put a null character in the first position of the field.

Call Buffer:

```
type  fmrec = record
      func    :    3        (One word)
      status  :    Returned (One word)

                   Bit 3: rfile contains a file name.

                   Bit 4: rdir contains a directory name.

      title   :    Banner   (80 bytes)
      appl    :    Function key label, packed array (8 bytes)
      type    :    Function key label, packed array (8 bytes)
      ddir    :    Directory to display, packed array (66 bytes)
      rfile   :    Returned file name (66 bytes)
      rdir    :    Returned directory (66 bytes)
end
```

THIS PAGE SHOULD BE BLANK

PROGRAMMING SPECIAL FEATURES

SECTION 7

PROGRAMMING DATA COMMUNICATIONS

The HP 150 comes with built-in, full-featured HP terminal capability. Users of your application can put the HP 150 in terminal mode by configuring it as a terminal via MS-DOS (see "Configuring the HP 150" in the *HP 150 Personal Computer Owner's Guide*). The user can also use P.A.M. to put the HP 150 in terminal mode (by touching the Terminal function key). There is no trivial way to put the HP 150 in 'terminal mode' programmatically.

Writing Your Own Data Comm Program

Unlike other computers that you may be accustomed to, Hewlett-Packard does not provide hardware port addresses (for example, the USART controller chip address). There is also no way to alter terminal characteristics, such as baud rate, programmatically.

The recommended, straightforward way to write your own data communications program is to use the data comm devices, COM1 and COM2. You open, read and write these devices using MS-DOS file calls. Characteristics for COM1 and COM2, such as baud rate, are configured by the user (see the next section).

Assigning the COM1 and COM2 Devices to Ports 1 and 2

The default device assignments for COM1 and COM2 are Ports 1 and 2, respectively. You can change these assignments as necessary by following the procedures in "Configuring MS-DOS" in the *HP 150 Personal Computer Owner's Guide*. You can reassign both COM1 and COM2 to any of these devices: Port1, Port2 Serial and Remote. *Remote* is typically a mainframe computer and is the "Remote/Serial Dev" entry you see when you configure your hardware (see "Configuring the HP 150" in the *HP 150 Personal Computer Owner's Guide*).

Note that each RS232 port has a 256 byte buffer.

Special I/O Control Functions

There are some additional functions that you can use to control data comm operations. You perform these operations by using the I/O Control Function call 44H. The following paragraphs describe the additional operations that are available:

SET 7-BIT MODE.

This function discards the eighth bit of each byte, since it is assumed to be a parity bit. This is the default mode of operation for data comm devices.

(8,1)	Function Code

SET 8-BIT MODE.

This function specifies that all eight bits be returned to your program. If handshaking is still active, some control codes may be interpreted and absorbed by the system. To obtain complete control, use this function in conjunction with the "Disable Handshaking" function which follows.

(8,2)	Function Code

DISABLE HANDSHAKING.

This function specifies that all characters be passed through to your program. The HP 150 performs none of the normal handshaking activities. You can specify this function independently of the 7-bit and 8-bit functions (the two previous functions).

(8,3)	Function Code

ENABLE HANDSHAKING.

This function puts the handshaking options specified for Port1 and Port2 in effect. Some control characters are interpreted/absorbed by the firmware.

(8,4)	Function Code

DISCONNECT MODEM.

This function drops the DTR signal which generally causes a modem disconnect. You may need to verify the configuration of the external modem.

(8,5)	Function Code

???

SEND BREAK.

This function is the equivalent of typing the [Break] key. It does this by holding the DTR line low for 200 milliseconds.

(8,6)	Function Code

FAST SEND.

This function transmits a block of data at a time rather than transmitting one byte at a time. It is similar to a normal MS-DOS write except that it is somewhat more efficient.

(8,7)	Function Code
(BUFFER)	The segment and offset address of the data buffer.
(DATA LENGTH)	This is the number of bytes to send.

Programming Considerations

When performing data comm operations, be sure to set 'Raw Mode' on. This is done with an I/O Control call, and is presented in the MS-DOS Programmers Reference Manual.

When you open a file, it is normally in 'cooked' mode. This means MS-DOS will process all characters accepted from the file/device to check for end-of-file (CTRL-Z), printer echo, and other special characters.

By setting 'raw' mode, you can prevent MS-DOS from this unnesasary processing and speed up the overall operation of data comm in the process.

KEYCODE MODES

What Is Keycode Mode?

For applications to obtain additional control over the HP 150 keyboard, the HP 150 AGIOS includes "Keycode Mode". By using this mode, your application can detect keypresses as well as the state of the five 'shift-type' keys: left and right [Shift] key; left and right [Extend char] keys; and the [CTRL] key. It also lets you distinguish between touchscreen and keyboard input.

To obtain this control requires no special interrupts or calls: all keycode activity is directed to standard console input.

Using Keycode Mode

To get started with keycode mode, you will need to do some set-up. The steps involved in this set-up are:

- Define Key Characteristics with AGIOS call (0,40)
- Turn on 'Raw Mode' with MS-DOS call 44 hex
- Turn off Control-C checking with MS-DOS call 33 hex
- Turn on keycode mode with AGIOS call (0,43)

When all this has been done, you will need to accept four bytes of data for each keypress: more will be said on this shortly (see Reading Keycode Data).

Key characteristics can be specified for any of the 'special' keys on the HP 150. Key characteristics include the ability to process a key normally, to 'intercept' a key, or to ignore it. You can also have the HP 150 'beep' when individual keys are pressed, in any combination with the other key characteristics. Refer to Section 5, AGIOS Programming for additional information.

Device files on MS-DOS have two processing modes: 'Cooked Mode' and 'Raw Mode'. In cooked mode, the default mode for devices, MS-DOS checks each character for special significance. For example, Control-C, Control-P, and Control-Z all have special meanings to MS-DOS. When a device is in raw mode, MS-DOS stops checking for thiese special characters. Raw mode generally allows faster operation on devices as well.

You can set and clear raw mode with a short assembler program. Sample subroutines which control raw mode at the console follow:

```
RAWON  MOV  AX,4400H      ;IO Ctrl Get Device Information
       MOV  BX,1          ;File handle: 1 for console
       INT  21H
       XOR  DH,DH         ;Clear High Byte for Put
       OR   DL,20H        ;Set Raw Bit
       MOV  AX,4401       ;IO Ctrl Put Device Information
       MOV  BX,1          ;File Handle: 1 for Console
       INT  21H
       RET                ;Done : Return to Caller

RAWOFF MOV  AX,4400H      ;IO Ctrl Get Device Information
       MOV  BX,1          ;File handle: 1 for console
       INT  21H
       XOR  DH,DH         ;Clear High Byte for Put
       AND  DL,0DFH       ;Clear Raw Bit
       MOV  AX,4401H      ;IO Ctrl Put Device Information
       MOV  BX,1          ;File Handle: 1 for Console
       INT  21H           ;
       RET                ;Done : Return to Caller
```

You will find this code on the ISV Development Disc in file RAWMODE.ASM.

Control-C trapping is normally handled by MS-DOS when I/O function calls occur. You want to verify that this is disabled by setting Control-C trapping off with MS-DOS function 33 hex.

Finally, turn on keycode mode with AGIOS call (0,43). This call is very simple:

```
SETKC   MOV   AX,4403H    ;IO Ctrl Write
        MOV   BX,1        ;Console Handle
        MOV   CX,3        ;Buf length
        MOV   DX,OFFSET BUF
        INT   21H
        RET
BUF     DB    43
        DB    0
        DB    1           ;1 -> ON; 0 -> OFF
```

Before your application ends, be sure to 'clean up'. This usually means setting all keys to their default characteristics, turning off keycode mode, enabling Control-C trapping, and setting 'cooked' mode on the console. This is good programming practice and means the next program to run will not have problems because of your application.

Reading Keycodes

Once keycode mode is enabled, each keypress generates four bytes: two bytes in a 'qualifier word' and two bytes in a 'data word'. The format of these four bytes is presented in Section 5, AGIOS Programming. However, here you'll see a more in-depth explanation.

Qualifier Word.

The two-byte qualifier word is in the following format:

```
+----------------------------+
|  Device Id  |  Shift Bits  |
+----------------------------+
```

The Devide Id field indicates which device was the source of the key. Possible values for Device Id are:

Device Id	Data Source
00 Hex	Internal Code
080 Hex	Touchscreen
0C0 Hex	Keyboard

The 'Internal Code' is not a device you should ever receive. It is used internally for communication between the terminal and computer firmware.

The Shift Bits field indicates additional information about the keypress. You can determine which, if any, shift keys were depressed;, whether the key is an ASCII key or a special key;, and whether this keypress is due to auto-repeating on the keyboard. The format of the Shift Bits byte is:

```
+------------------------------------+
| S : R : LE : RE : CT : LS : RS : RP |
+------------------------------------+
```

The meaning of each bit follows:

S:	Special Bit. This bit set indicates the key is one of the 'special' keys and the Data Word contains a keycode from the table of keycodes. When this bit is cleared, the key is an ASCII key and the Data Word contains an ASCII code.
R:	Reserved Bit. This bit is reserved for future expansion.
LE: RE:	These bits indicate the state of the left (LE) and right (RE) [Extend char] key. These keys normally shift into additional 8 bit ASCII characters.
CT:	This bit indicates the control key [CTRL] was pressed.
LS: RS:	These bits indicate the state of the left (LS) and right (RS) shift keys. Note that when the Special Bit is cleared, the ASCII code in Data Word will correspond to upper or lower case ASCII codes independent of the state of these bits. That is, you don't need to check these bits to see if a character is upper case or not: the proper ASCII code is returned to you in Data Word.
RP:	This bit indicates that the device is auto-repeating and this key is a result of a previous key which has been held down. If auto-repeating keys are not acceptable to your application, you can check this bit and ignore the keypress if the bit was set.

Data Word.

The format of the Data Word follows:

```
+----------------------+
|   Null   |   Data    |
+----------------------+
```

The high order byte (Null)is null and will be all zeros.

The low order byte (Data) contains an ASCII code if the Special Bit was cleared or a Keycode value if the Special Bit was set.

Note that, if the Data is ASCII, the case will correspond to the proper upper or lower case ASCII code. There is no need to check the state of the Shift Bits.

Sequence of Data.

When in keycode mode, the four bytes will be sent in the following sequence:

```
Shift Bits      <- First byte
Device Id
Data Byte
Null Byte       <- Last byte
```

Note that some languages ignore null input. BASIC is an example of such a language, and you will not be able to perform keycode input in these languages. It is suggested you use an assembler input routine to accept this data. You may further wish to use an input function which does not process Control-C. MS-DOS function 7 is an example of such a function.

Touch Screen Input In Keycode Mode

In keycode mode, the touchscreen will return reports in the same format as when not in keycode mode. However, each character will return four bytes (`Qualifier Word` and `Data Word`). Let's review each type of field and row/column reports.

`ASCII Fields`. Each character in the ASCII buffer will will be reported as four characters. This is another good reason to keep field lengths to a minimum!

`Keycode Fields`. This type of field allows you to specify a `Qualifier Word` and `Data Word`. You will receive

just those four bytes on return.

`Toggle and Normal Fields`. These two fields each return three character reports in normal mode. In keycode mode, you will receive four bytes for each of these three characters. In other words, you will receive 12 bytes of data for each report in Toggle and Normal fields.

`Row/Column Reports`. In normal mode, you normally receive three chartacter row/column reports. When keycode mode is on, you will receive four bytes for each of these, which means you must process 12 characters for each report.

Softkeys in Keycode Mode

The User Defined Softkeys are reported as a buffer of text and are defined with the `ESC & f` escape sequence. You will find that, like ASCII Touch Fields, you receive four bytes for each character in the User Defined Softkey buffer.

If you are using Applications Softkeys (from AGIOS), you will receive only a four byte report. Note that Applications Keys will set the `Special Bit`, and the `Data` represents a keycode.

GRAPHICS TEXT

Introduction

The HP 150 features a full graphics character set implemented in ROM. With graphics text, you can specify such attributes as slant, size, and orientation of characters sent to standard console output.

You can also create your own custom characters, either one at a time or as a replacement for the entire set stored in the HP 150. This section will discuss how to accomplish both of these objectives.

Creating a Graphics Character

First, let's take a look at the format of a single graphics text cell. Remember, a graphics character cell is 7 pixels by 8 pixels.

```
 (0,6)  +--------------+  (7,6)
        |              |          Note that (0,0) is on the base line,
        |              |          NOT at the lower left of the cell.
        |              |
        |              |
        |              |
_(0,0) _|_ _ _ _ _ _ _ | _ _ _ _ _   Base Line
        |              |
        |              |
 (0,-2) +--------------+  (7,-2)
```

As you will see, cell coordinates are specified in "excess 64's"
notation. Basically this means you need to add 64 decimal to the coordinates you see above.

Graphics characters are stored as a vector list describing how the character is formed. Within this list, you must 'describe' the character within a cell. The vector list format is:

```
+----------+
|  X L L   |     Coordinates for the lower left and
|----------|
|  Y L L   |     upper right points of this character
|----------|
|  X U R   |     used in scaling the character within
|----------|
|  Y U R   |     the character grid.
|----------|
|   X 1    |     Ordered pairs or (X,Y) points which
|----------|
|   Y 1    |     describe the character. These are
|----------|
|   X 2    |     described in detail below.
|----------|
|   Y 2    |
|----------|
-          -
|----------|
|   X n    |     The last (X,Y) must be (0,0) entries.
|----------|
|   Y n    |
+----------+
```

Each value in the above vector list is stored in 'excess 64's notation', which means you must add 64 to the relative coordinate within each grid.

The Vector List

In the vector list illustrated above, the first four entries describe the relative cell size of the character. These four bytes specify the two coordinates of the lower left pixel and the upper right pixel. By combining this information with the character cell illustrated above, we can create a character cell in excess 64's notation. This character cell is:

```
(64,70) +--------------+ (71,70)
        |              |
        |              |
        |              |
        |              |
        |              |
(64,64)_|_ _ _ _ _ _ _ | _ _ _ _ _   Base Line
        |              |
        |              |
(64,62) +--------------+ (71,62)
```

Given this 'standard size cell', the first four entries in a vector list describing a character would be:

```
+----------+
|    64    |     - >  Lower left X coordinate
|----------|
|    62    |     - >  Lower left Y coordinate
|----------|
|    71    |     - >  Upper right X coordinate
|----------|
|    70    |     - >  Upper right Y coordinate
+----------+
```

After these first four bytes, the vector list specifies an (X,Y) coordinate pari for each endpoint in the character. The X coordinate represents the X value and the Y coordinate represents the Y value, both in excess 64's notation. The X coordinate also specifies one additional bit of data: whether to 'move' to this point or to 'draw' to the point.

If the most significant bit of the X coordinate is zero, the (X,Y) pair specifies a 'Draw' mode. The 'pen' draws to the (X,Y) point. If the most significant bit is one, the (X,Y) pair indicates a 'Move' mode. The 'pen' is 'lifted' and 'moved' to (X,Y).

Let's take a look at an example of building a character given the information you now have.

Building a Character

Here is a single character drawing of the special character we want to create. It is presented 'in' the character grid from above.

```
(64,70) *--------------* (71,70)
        | *          * |
        |   *      *  *|
        |     *  *    *|
        |       *     *|   < - The intersection here
        |     *       *|       occurs at (66,68)
(64,64)_|_ _*_ _ _ _ _*|  _ _ _ _ _     Base Line
        |  *          *|
        | *           *|
(64,62) *--------------+ (71,62)
```

Prior to building a vector list, let's describe 'how' to draw the character.

- Start at the lower left by 'moving' to (64,62). Remember, this is the 'cell' coordinate (0,-2) expressed in excess 64's notation.
- Draw to (71,70). This is the upper left, cell coordinate (7,6).
- Draw to (71,62).
- Move to the upper left, (64,70).
- Draw to (66,68).

Having defined the steps above let's write the code for the above character.

```
    BUF   DB     64                ;X Lower Left
          DB     62                ;Y Lower Left
          DB     71                ;X Upper Right
          DB     70                ;Y Upper Right
; Note the + 128 indicates setting the 'move' bit
          DB     64 + 128          ;Move to X: 64
          DB     62                ;        Y: 62
          DB     71 + 0            ;Draw to X: 71
          DB     70                ;        Y: 70
          DB     71 + 0            ;Draw to X: 71
          DB     62                ;        Y: 62
          DB     64 + 128          ;Move to X: 64
          DB     70                ;        Y: 70
          DB     62 + 0            ;Draw to X: 62
          DB     66                ;        Y: 66
          DB     0                 ;Last X
          DB     0                 ;Last Y
```

Once you've built this list, you can use AGIOS function (4,37), 'Output Single Character'. Specify the offset to BUF above, and the character will be sent to the screen using the orientation, size, and slant characteristics which apply to the standard graphics text.

Redefining the Entire Character Set

The HP 150 character set is defined by a list of 255 entries. The contents of this list are 16 bit pointers to character vector lists as described above. In fact, the entry into the pointers list is the ASCII code of the character. For example, the 65th entry in the list of pointers corresponds to ASCII code 65 decimal, which is the character 'A'. That 65th entry points to the head of the graphics vector list for 'A'.

To re-define the HP 150 character set, you need to build a list of pointers to take the place of the ROM-based list. This entails creating your own list of 255 16 bit pointers.

The first 32 entries in the list of pointers correspond to ASCII values 0 through 31. These characters are not part of the HP 150 graphics character set but they must exist and contain zero. The same is true of ASCII values 128 through 159.

The 32nd character, ASCII 'Space', is an important entry in the list. The system uses the (XLL,YLL) and (XUR,YUR) values to 'size' the entire set. For example, to define a single-cell sized character set, similar to the ROM-based set, the vector list pointed to by the 32nd entry must be:

```
    SP    DB     64                ;XLL Coordinate
          DB     62                ;YLL Coordinate
          DB     71                ;XUR Coordinate
          DB     70                ;YUR Coordinate
          DB     0                 ;Last X Byte
          DB     0                 ;Last Y Byte
```

The remaining entries in the list of pointers simply contain offset addresses (16 bit) to the character which re-defines the ASCII character with that code. However, remember that the offset is *from the start of the list*, not an absolute offset.

Once you have defined such a table, AGIOS call (4,35) is used to re-define the graphics character set. In that call, you specify the offset and segment of the first entry in your list of pointers. (Remember: this first entry represents ASCII code 0. You must have 31 null pointers at the top of your list, and again in entries 128 through 159).

Programming Considerations

When creating graphics characters, the value of (XUR,YUR) does not always need to be set to the upper right coordinate of the character cell. For example, to implement 'proportional spacing' within graphics text, you might define some characters to be wider and others narrower than the standard cell size.

In fact, the values of (XLL,YLL) and (XUR,YUR) are arbitary with respect to the character cell. If you want a character to be two cells high, for example, simply specify (XUR,YUR) of (14,14). (That's right: check the numbers yourself!).

Likewise, you can also move (XLL,YLL) below the lower coordinates of the cell. Either way, be sure to specify the appropriate values in the first four bytes of the vector list!

Finally, if you replace the entire character set, remember that the offset stored in the list of offsets must specify an offset *from the start of the table, not an absolute offset!*

AGIOS AND ESCAPE SEQUENCE QUICK REFERENCE

This Appendix is intended to give you a quick reference to the AGIOS function calls and the escape sequence equivalents, where applicable. You need to refer to Section 2 to determine the parameters values which correspond to each of these function calls.

AGIOS Call	Description	Equivalent Escape Sequence
0,0	Batch	
AIOS - Video Intrinsics		
0,1	Define Area	
0,2	Write Area	
0,3	Clear Area	
0,4	Enhance Area	
0,5	Read Area	
0,6	Shift Area	
0,7	Write Line	
AIOS - Application Softkeys		
0,8	Update Softkey Label	
0,9	Read Softkey Label	
0,11	Display Softkey Labels	
AIOS - Control Functions		
0,16	Execute Two Char. Sequence	ESC char
0,17	Position Cursor	ESC & a
0,18	Define Enhancements	ESC & d
0,19	Cursor Sense Absolute	ESC a
0,20	Cursor Sense Relative	ESC
0,21	Set Cursor Type	
0,22	Read Cursor Type	
0,24	Read Terminal Configuration	
AIOS - Touch Screen Functions		
0,32	Define Touch Field	ESC - z g
0,33	Define Softkey Field	ESC - z s
0,34	Delete Touch Field	ESC - z d
0,35	Touch Screen Reset	ESC - z j
0,36	Set Touch Sensing Modes	ESC - z n

AGIOS Call	Description	Equivalent Escape Sequence
AIOS - Keyboard Intercept		
0,40	Define Key Characteristics	
0,41	Get Key Status	
0,42	Put Key	
0,43	Keycode ON/OFF	
0,44	Keycode Status	
0,45	Read Keypad Status	
GIOS - Display Control		
4,1	Clear Graphics Memory	ESC * d a
4,2	Set Graphics Memory	ESC * d b
4,3	Turn On Graphics Display	ESC * d c
4,4	Turn Off Graphics Display	ESC * d d
4,5	Turn On Alphanumeric Display	ESC * d e
4,6	Turn Off Alphanum. Display	ESC * d f
4,7	Turn On Graphics Cursor	ESC * d k
4,8	Turn Off Graphics Cursor	ESC * d l
4,9	Turn On Rubber Band Line	ESC * d m
4,10	Turn Off Rubber Band Line	ESC * d n
4,11	Move Graphics Cursor Abs.	ESC * d <x,y> o
4,12	Move Graphics Cursor Rel.	ESC * d <x,y> p
4,13	Turn On Alphanumeric Cursor	ESC * d q
4,14	Turn Off Alphanumeric Cursor	ESC * d r
4,15	Turn On Graphics Text Mode	ESC * d s
4,16	Turn Off Graphics Text Mode	ESC * d t
GIOS - Vector Drawing Mode		
4,17	Select Drawing Mode	ESC * m <mode> a
4,18	Select Line Type	ESC * m <type> b
4,19	Define Line Pattern/Scale	ESC * m <pattern> <scale> c
4,20	Define Area Fill Pattern	ESC * m <pattern> d
4,21	Fill Rectangular Area, Abs.	ESC * m <x1,y1,x2,y2> e
4,22	Fill Rectangular Area, Relocatable	ESC * m <x1,y1,x2,y2> f
4,23	Select Polygonal Fill Pattern	ESC * m <pattern> g
4,24	Select Boundary Pen	ESC * m <pen> h
4,25	No Polygon Boundary	ESC * m h
4,26	Set Relocatable Origin	ESC * m <x,y> j
4,27	Set Relocatable Origin to Pen Position	ESC * m k

AGIOS Call	Description	Equivalent Escape Sequence
4,28	Set Relocatable Origin to Cursor Position	ESC * m l
4,29	Set Graphics Text Size	ESC * m <size> m
4,30	Set Graphics Txt Orientation	ESC * m
4,31	Turn On Text Slant	ESC * m o
4,32	Turn Off Text Slant	ESC * m p
4,33	Set Graphics Text Origin	ESC * m <0-9> q
4,34	Graphics Text Label	ESC * l <text>
4,35	Define User Character Set	
4,36	Select Default Character Set	
4,37	Output Single Text Character	
4,38	Set Graphics Default	ESC * m r
4,72	Set Picture Def. Defaults	ESC * m l r
4,73	Graphics Hard Reset	ESC * w r

GIOS - Graphics Plotting

4,9	Lift Pen	ESC * p a
	Vector Move	ESC * p a <x,y>
4,0	absolute	
4,1	incremental	
4,2	relocatable	
4,3	Lower Pen	ESC * p b
	Vector Draw	ESC * p b <x,y>
4,4	absolute	
4,4	incremental	
4,4	relocatable	
4,4	Set Pen Position to Cursor Position	ESC * p c
4,4	Point Plot	ESC * p d
4,4	Set Relocatable Origin to Pen Position	ESC * p e
4,5	Start Polygonal Area Fill	ESC * p s
4,5	Term. Polygonal Area Fill	ESC * p t
	Polygon Move:	
4,52	absolute	
4,53	incremental	
4,54	relocatable	
	Polygon Draw:	
4,55	absolute	
4,56	incremental	
4,57	relocatable	
4,58	Lift Boundary Pen	ESC * p u
4,59	Lower Boundary Pen	ESC * p v

AGIOS Call	Description	Equivalent Escape Sequence
GIOS - Graphics Status		
4,60	Read Device Id	ESC * s 1 ^
4,61	Read Pen Position	ESC * s 2 ^
4,62	Read Cursor Position	ESC * s 3 ^
4,63	Read Cursor Position, Wait For Key	ESC * s 4 ^
4,64	Read Display Size	ESC * s 5 ^
4,65	Read Graphics Capability	ESC * s 6 ^
4,66	Read Graphics Text Status	ESC * s 7 ^
4,67	Read Zoom Status	ESC * s 8 ^
4,68	Read Relocatable Origin	ESC * s 9 ^
4,69	Read Reset Status	ESC * s 10 ^
4,70	Read Area Shading	ESC * s 11 ^
4,71	Read Dynamics	ESC * s 12 ^
4,74	Read Extended Screen Dimensions	

SAMPLE PROGRAMS

APPENDIX B

AGIOS FUNCTION CALLS FROM A HIGH LEVEL LANGUAGE

The purpose of this appendix section is to give you examples of how to use the AGIOS function calls in high-level languages. The language examples included here is in MS-Pascal. The 8086/8088 assembly language and MS-DOS system calls are included in this section also since they are needed when you use AGIOS functions.

GETTING STARTED WITH PASCAL

To use AGIOS function calls, you need a working knowledge of the following areas:

- HP 150 capabilities (touch screen, graphics, etc.)
- AGIOS function calls
- Intel 8086/8088 assembly language
- MS-DOS system calls
- MS-Pascal

You need these reference materials handy:

- The MS-Pascal language manual
- The MS-Pascal compiler manual
- The MS-DOS programmer's reference manual
- An 8086/8088 assemby language reference manual

You should have the following hardware and software:

- The MS Pascal compiler and linker
- The MS 8088 macro assembler
- A text editor or word processor to create source code files

What the Pascal Program is Going to Do

The purpose of the Pascal routine in this section is to show you how to home the cursor. Even though the operation is a simple one, you can apply the concepts to do any task you need to perform. The Pascal program calls an external assembly language routine. The assembly language routine does the following:

- Loads various memory registers with the parameters of the operation to be performed, in this case, homing the cursor.
- Issues an interrupt instruction which passes control to MS-DOS.
- After control is returned to the assembly language routine, it checks the error status of the completed AGIOS call. This error status is contained in the AX register (For other AGIOS functions, information may be loaded into predefined buffer locations. This lets Pascal receive data from the AGIOS call).
- The assembly language routine passes control back to the Pascal program.

Passing AGIOS Inputs From Pascal

To pass the input information to your assembly language routine you need to create a buffer. This buffer contains all the required input information and should be passed by reference. In addition, you need to tell the assembly language routine the length of the buffer. The buffer length field is an integer field that is passed by value.

The first thing you need to do is to create a data buffer that contains the AGIOS function call code and other information that may be required for the AGIOS function call that you're going to perform. The example which follows is going to home the cursor by using the AGIOS call, "Execute Two Character Sequence". The buffer is set up as a PACKED ARRAY OF CHAR, with the value 0 in byte 1, the value 16, in byte 2, and the character H in byte 3. However, because the 8088 chip addresses information in a word in reverse order (high byte then low byte), we actually need to pass the data as follows:

```
                 +-----------------+
    Contents--> | 16   |  0  |  H  |
                 +-----------------+
    Byte    -->    0     +1    +2
```

Since the data buffer is three bytes, we need to pass the length 3 to the assembly language program.

Here is an example of a simple Pascal program that homes the cursor:

```
program try_agios (input,output);

        {This program demonstrates the passing of input
        parameters to an assembly language routine.  The program
        homes the cursor.}

        {define a agios data buffer type}

        type  buffer = record
              low_byte  : byte;
              high_byte : byte;
              char_code : char
        end;

        {declare assembly language routine as external; pass the
        data by reference and the length by value.  Pick up the
        AGIOS error status as an integer result of the function
        (0 = no error).}

        function agios( var  parms : buffer;
                        length : integer) : integer; extern;

        {declare variables}

        var  homeup : buffer;
             ret    : integer;

        begin

           {load the agios input values into the buffer}

           homeup.low_byte := 16;
           homeup.high_byte := 0;
           homeup.char_code := 'H';

           {execute agios homeup function and get error status}

           ret    := agios(homeup,3);

           {write return status; zero is no error}

           writeln('return status: ',error);

         end.
```

NOTE

MS-Pascal has the predefined type BYTE. See the MS-Pascal language reference manual for more information.

It may seem like a lot of work to just home the cursor, but the AGIOS method provides faster I/O. There are advantages to using more powerful AGIOS functions that write large blocks of text to the screen and handle graphics output.

Assembly Language Interface to Pascal

When the Pascal program passes control to the assembly language routine, that routine must load certain registers so that AGIOS can perform the requested function. The last step in the assembly language routine is to issue a *INT 21H* call. This call actually passes control to MS-DOS to perform the function. The following registers are used when you issue an AGIOS call:

REGISTER:	FUNCTION:
AH	Contains the MS-DOS system function number that does device I/O control. This is function number 44H and is described in the MS-DOS programmer's reference manual.
AL	Contains a function request number that specifies which of the I/O control options one wants to use in function 44H. For AGIOS calls, this option number is 3 and means that you're going to write to the device control channel.
BX	Contains the device handle number. The input functions of the console are assigned a handle of 0. The output console functions are assigned 1. All AGIOS functions use 1 as the device handle.
CX	Contains the number of bytes being passed in the input buffer. In the earlier "home cursor" example, this was 3.
DS	Contains the address of the segment that the input data is in. This address is passed to the routine when we assign the data buffer the characteristic VAR which means the data will be passed by reference (address).
DX	Contains the offset of the input data buffer from the segment address. This is part of the address passed when we pass data by reference.

Upon return from the interrupt, the AX register is loaded with an error status. Zero indicates no error occurred while non-zero values indicate that an error did occur. The Pascal example which follows shows the AX being returned.

You should read the section in the MS Pascal compiler manual that discusses passing parameters. Here are a few key points to remember:

- Pascal only does FAR calls. Each call puts a segment address and a segment offset on the stack.
- Pascal loads the passed parameters on the stack from left to right. Some languages like Basic and C do it in the opposite direction.
- Before you begin executing the main body of assembly language rountine, you should save the values of the BP and DS registers by pushing them on the stack. At the end of the routine you should restore these registers to their initial value. Why? -- because the compiler manual says so!
- Clean up space allocated on the stack by using the assembler instruction RET <n>. <n> represents the number of arguments which is 2, in this case (buffer and length).

Here is an example of an assembler routine that will handle AGIOS output functions:

```
pgroup     group      prog
prog       segment    byte public 'prog'
           assume     cs:pgroup

           public     agios
;                ***  CODE ***
;   calling convention:  AGIOS(buf,count)
;
;   picture of stack after call, before execution of this
;   assembler procedure:
;
;     high memory
;          +----------------------+
;          |low byte  | high byte| Address of agios data
;          +----------------------+
;          |low byte  | high byte| Buffer length value
;          +----------------------+
;          |low byte  | high byte| Return segment address
;          +----------------------+
;     SP-> |low byte  | high byte| Return offset address
;          +----------------------+
;     low memory
;                     stack grows down
;
;
agios      proc       far
push       bp               ;save callers registers
push       ds
mov        bp,sp            ;move stack pointer in bp
mov        dx,[bp+10]       ;get buffer address
mov        cx,[bp+8]        ;get buffer length
mov        bx,1             ;console output handle

mov        ax,4403H         ;call the I/O control function
int        21H              ;44H and use option 3 to write
                            ;to device control channel

pop        ds               ;restore registers
pop        bp

ret        4                ;return 4 bytes, previously
                            ;occupied by agios parms to
                            ;stack.
agios      endp
prog       ends
           end
```

Passing Pointers to AGIOS: Pascal

Some AGIOS functions require blocks of data that are not in the input buffer to be passed to or from your program. For example, the WRITE AREA function needs a double word pointer to the buffer which contains the text that you wish to write to the screen. The READ AREA function requires a double word pointer to a buffer which AGIOS uses.

The double word pointer has the following structure:

word 1 - 16 bit offset address of the data buffer
word 2 - 16 bit segment value

The best way to get this information is to use a special data type available in MS-Pascal, ADS. ADS provides exactly the information you need to get the buffers address.

The following example uses the AGIOS function which returns the absolute position of the alpha cursor. The AGIOS input string for this function is the address of the buffer to be used for storing the cursor position.

```
program cursor_sense_absolute (input,output);

type return_buf = array[1..2] of integer;
      input_buf = record
      func_low_byte : byte;
      func_hi_byte  : byte;
      address       : address of return_buf
end;

var  ret            : integer;
     agios_inputs : input_buf;
     cursor_pos   : return_buf;
```

```
function agios ( var parms : input_buf;
                 length : integer) : integer; extern;

begin

{load function code 0,19}
agios_inputs.func_low_byte := 19;
agios_inputs.func_hi_byte  := 0;

{load address of output buffer}
agios_inputs.address := ads cursor_pos;

{call agios function}
ret   := agios(agios_inputs,6);
{cursor position is now in the integer array CURSOR_POS}

writeln('return    status    =',ret);    writeln('row     =',cursor_pos[1]);
writeln('col =',cursor_pos[2]);

end.
```

Refer to your MS-Pascal language reference manual for more information on using the ADS type and function.

THE HP 150 KEYBOARD

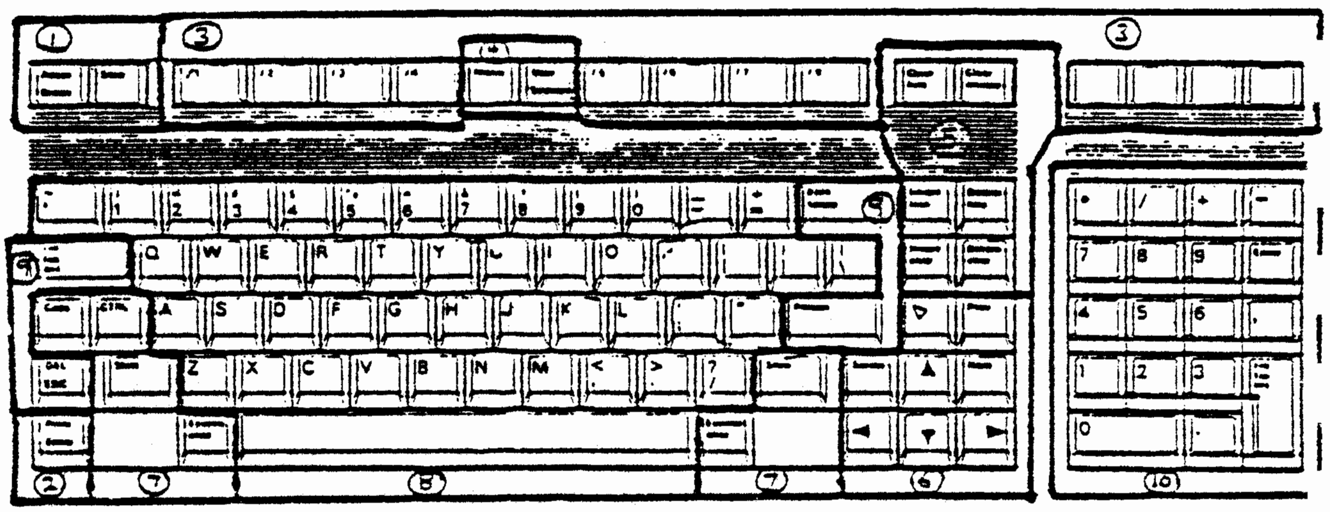
APPENDIX C

This Appendix describes the HP 150 keyboard giving details about the various keys and key combinations. It is intended to be a supplement to the *HP 150 Terminal User's Guide.*

The key descriptions in this appendix are listed by groups, depending on their location on the keyboard. Figure 1 gives you a quick reference to the groups discussed.

The key descriptions given in this appendix are actually the firmware interpretations. That is, these meanings are the default meanings to the HP 150. Your application program can intercept these keys, and inhibit or alter their meanings.

FIGURE 1: THE HP 150 KEYBOARD - KEY GROUPS



STOP/RESET GROUP (1)

[Reset Break]

This key can be used the following ways:

1. [Reset Break]

   A data comm break signal is sent to the remote data comm. This key is interceptable via keycode mode.

2. [Shift] [Reset Break]

   Causes a soft reset of the local hardware to occur. This key is not interceptable.

3. [CTRL] [Shift] [Reset Break]

   Causes a hard reset of the local hardware to occur. This key is not interceptable.

[Stop]

This key can be used the following ways:

1. [Stop]

   Toggles between starting and stopping the system. This is accomplished by causing the HP150 to enter a loop awaiting a STOP or HARD RESET. This key can be intercepted. However, it is recommended that applications enable this function.

2. [Shift] [Stop]

   In non-terminal mode, this key has no effect.

   In terminal mode, with no calling program to return to, the HP150 attempts to boot MS-DOS. This key can be intercepted using keycode mode.

3. [CTRL] [Stop]

   This key functions identically to [Shift] [Stop].

PRINT ENTER GROUP (2)

[Print Enter]

This key can be used the following ways:

1. [Print Enter]

   This key functions identically to [ENTER] on an HP262X terminal. In block mode, [ENTER] can be strapped for line or page transmission using no handshaking, short or long handshaking. You can make [Return] function like [ENTER] via the terminal configuration menu. This key is interceptable.

2. [Shift] [Print Enter]

   This key acts as an "alpha print" key. Its function is identical to a home up followed by a copy all (to the current destination device). This key is interceptable.

3. [CTRL] [Print Enter]

   This key combination acts as a "graphics print" key. Its function is to do a raster dump of graphics memory to the "to device". This key is interceptable.

FUNCTION KEYS GROUP (3)

There are three types of function keys on the HP 150:

System Softkeys - are implemented by the firmware and perform various system functions. These keys may be locked out by an application.

User-Definable Softkeys - are the set of 8 function keys that can be programmed via escape sequences. You can define a string of up to 80 characters and a label of two sets of 8 characters via a menu (accessible using the [CTRL] [Menu] key combination). These keys cannot be intercepted.

Application Softkeys - return keycodes as listed in the "Keycode Table". The first eight keys have labels which can be defined programmatically via AIOS calls. The next four keys have no on-screen label but can be accessed the same way. These keys only work via keycodes intercepted by an application program.

FUNCTION KEY CONTROL GROUP (4)

The following terms are used in the key definitions for this group:

AIDS - The top of the "terminal" softkey tree. From AIDS, the user can get to other levels of the "terminal" softkey tree including MODES.

USER-DEFINED SOFTKEYS - The user-defined softkeys are currently defined in the HP262x terminals. The [CTRL] [MENU] key displays the menu for the user to define these keys in the HP 150. Esc&f sequences also define these keys.

APPLICATION SOFTKEYS - These keys have labels which are defined by the application via the AGIOS. There are no associated definition strings. The user cannot define these labels.

[Menu]

This key can be used the following ways:

1. [Menu]

   Toggles the display of softkeys on and off. Typing [User System], [Shift] [User System], and [CTRL] [User System] forces the display of softkeys on. The function keys are still active even when their labels are not displayed. This key is interceptable.

2. [Shift] [Menu]

   This key turns off the current screen labels and is interceptable.

3. [CTRL] [Menu]

   These keys display the user-definable softkeys menu. This key is not interceptable but can be locked out.

4. [CTRL] [Shift] [Menu]

   Enables or disables the entire touch screen (toggle). Applications can sense and toggle the status of this flag, however the key is not interceptable.

[User System]

This key can be used the following ways:

1. [User System]

   Displays the "aids" level of softkey labels. This application is interceptable (for example, the MODES level).

2. [Shift] [User System] This key enables and displays the application defined softkeys. It is not interceptable but can be locked out.

3. [CTRL] [User System]

   This key enables and displays the user defined or "typing aids" softkeys. It is not interceptable but can be locked out.

LOCAL EDITING GROUP (5)

All of the keys in this group are interceptable.

1. [Insert char]

   Toggles between INSERT and OVERSTRIKE modes. Displays the current status on line 27.

2. [Shift] [Insert Char] (with Wrap)

   Sets INSERT-WITH-WRAP mode; if INSERT is on, sets OVER-STRIKE mode. Displays the current status on line 27.

3. [Delete char]

   Deletes on character at the cursor position; the line is shortened by one character.

4. [Shift] [Delete char] (with Wrap)

   Deletes one character at the current cursor position and replaces the last character of the line with the first character of the next line. The next line is shortened by one character.

5. [Insert line]

   Inserts one line above the line that the cursor is on. Subsequent lines are moved down one line in the display.

6. [Delete line]

   Deletes the line that the cursor is on. Subsequent lines are moved up one line on the display.

7. [Clear line]

   Clears the current line from the cursor position to end of line (EOL).

8. [Shift] [Clear line]

   Clears the logical line regardless of the cursor position. This is the equivalent of ESC G followed by [Clear line].

9. [Clear display]

   Clears the display workspace from the cursor position to the end of the workspace.

10. [Shift] [Clear display]

    Clears the entire display workspace. Equivalent to a Home Up followed by [Clear display].

ALPHA CURSOR MOVEMENT (6)

All the keys in this group are interceptable.

1. [ ] (up arrow)

   Moves the alpha cursor up one line in the same column. When the cursor reaches the top line, "Up Arrow" rolls around to line 24.

2. [ ] (down arrow)

   Moves the alpha cursor down one line in the same column. When the cursor reaches the bottom line, "Down Arrow" rolls around to line 1.

3. [ ] (left arrow)

   Moves the alpha cursor left one character cell in the same row until the left edge of the physical window in reached. When the cursor reaches the left margin, the next "Left Arrow" rolls around to column 80 on the previous line.

4. [ ] (right arrow)

   Moves the alpha cursor right one character cell in the same row until the right edge of the physical window is reached. When the cursor reaches the right margin, the next "Right Arrow" rolls back to column 1 on the next line 1.

5. [Select]

   This key lets a user select the item to which the cursor is pointing.

6. [ ] (Home up)

   Moves the cursor to the first chracter of the workspace. Scrolls down if necessary to ensure that the cursor is in the display window. If memory lock is on, the cursor is positioned to column 1 on the first line below menu lock.

7. [Shift] [ ] (Home down)

   Moves the cursor to the bottom of the workspace. Scrolls up if necessary to ensure that the cursor is in the display window.

ALPHA SCROLL (6)

All of the keys in this group are interceptable.

1. [Shift] [ ] (Scroll down)

   Moves the text down in the window one line at a time until the top of the workspace is at the top of the window. The cursor does not move in the display window.

2. [Shift] [ ] (Scroll up)

   Moves the text up in the window one line at a time until the end of the workspace is reached. The cursor does not move in the display window.

3. [Shift] [ ] (Scroll left)

   This key performs no action in the HP 150 firmware.

4. [Shift] [ ] (Scroll right)

   This key performs no action in the HP 150 firmware.

5. [Next]

   Moves the text up in the window so the line immediately following the last line in the physical window becomes the new first line. [Next] always leaves at least one defined (though possibly blank) line in the window. The cursor is always placed in column 1, row 1 of the display window.

6. [Prev]

   Moves the text down in the window so the line immediately preceeding the first line in the physical window becomes the next last line. If there are not as many lines preceeding the window as needed to fill the window, [Prev] places the first line of the workspace at the top of the window. The cursor is always placed in colunm 1, row 1 of the display window.

GRAPHICS CURSOR MOVEMENT (6)

All of the keys in this group are interceptable.

1. [CTRL] [ ] or, (Graphics cursor up)
   [5] on the Graphics pad

   Moves the graphics cursor up one pixel. If the cursor is at the top of the graphics display, no action is performed.

2. [CTRL] [ ] or, (Graphics cursor down)
   [2] on the Graphics pad

   Moves the graphics cursor down one pixel. If the cursor is at the bottom of the graphics display, no action is performed.

3. [CTRL] [ ] or, (Graphics cursor left)
   [1] on the Graphics pad

   Moves the graphics cursor left one pixel. If the cursor is at the left edge of the graphics display, no action is performed.

4. [CTRL] [ ] or, (Graphics cursor right)
   [3] on the Graphics pad

   Moves the graphics cursor right one pixel. If the cursor is at the right edge of the graphics display, no action is performed.

5. [CTRL] [ ] (Graphics Home Up)

   Moves the graphics cursor to the top of the graphics display (upper left corner and displays the contents of the top of the graphics memory.

6. [CTRL] [Shift] [ ] (Graphics Home Down)

   Moves the graphics cursor to the bottom of the graphics display (lower left corner) and displays the contents of the bottom of the graphics memory.

7. [0] on the Graphics pad (Cursor Fast)

   Speeds up the graphics cursor motion if held down in conjunction with the graphics cursor-motion keys.

GRAPHICS SCROLL (6)

The [CTRL] [Shift] [arrow keys] are not implemented in the HP150 firmware. Unique keycodes are not defined, but an application wishing to provide this function can test for the [Shift] qualifier when a Graphics Cursor Motion keycode is received.

ANCILLARY GRAPHICS FUNCTIONS (5,10)

All keys in this group are interceptable.

1. [CTRL] [Clear display] (Graphics Clear)

   Clears all pixels in graphics memory.

2. [CTRL] [Insert char] or, (Alpha Display)
   [*] on the Graphics pad

   Toggles the display of the alpha memory

3. [CTRL] [Delete char] or, (Graphics Display)
   [/] on the Graphics pad

   Toggles the display of the graphics memory.

4. [CTRL] [Next] or, (Graphics Cursor Display)
   [+] on the Graphics pad

   Toggles the display of the graphics cursor.

5. [CTRL] [-] on Graphics pad (Graphics Pad Toggle)

   Toggles between NUMERIC PAD and GRAPHICS PAD modes. This key may be intercepted; an application can force the mode to NUMERIC and then intercept this key to ensure unique keycodes from this keypad. An indicator in line 27 shows the current status.

MODIFIER KEYS (7)

1. [CTRL]: Qualifier bit 3

   This key produces control codes in the base (C0) set only, never the C1 codes 129 - 159 (decimal). If [Extend char] and [CTRL] are used together, the character which would have been generated by [Extend char] alone will be converted to a control character from column 0 or 1 by zeroing the three high-order bits.

2. [Shift]: Qualifier bit 2 (left) or 1 (right)

   This key produces altered ASCII codes or keycodes as implied by the keyboard markings for graphic characters, and produces altered functions or keycodes for other keys as described in this document.

3. [Extend char]: No effect on qualifier byte

   A qualifier key which obtains the characters from the ROMAN8 (8th-bit set) character set, and displaced USASCII characters. See chart for key mappings. This key only affects keys in the "Main Touch Typing Group (8)".

4) [Shift] [Extend char]

   Obtains additional graphics characters from the ROMAN8 character set and displaced USASCII characters.

MAIN TOUCH TYPING GROUP (8)

These keys all send appropriate ASCII codes with or without [Shift]. When used with [CTRL], they send a C0 control character derived by clearing the three highest order bits of the character which would have been sent without [CTRL].

CONTROL CHARACTER KEY (9)

These keys send ASCII codes if not intercepted, and unique keycodes if inter- cepted.

1. [Tab]

   This key sends an ASCII HT (Hex 09) if not intercepted.

2. [Shift] [Tab] (Back tab)

   This key combination does not send an ASCII character (since there is no Back tab in ASCII), but moves the cursor to the previous tab stop if not intercepted. Note that if intercepted this key sends a keycode which is different from the keycode sent by [Tab] alone, in contrast to Numeric pad [Tab].

3. [DEL ESC]

   This key sends an ASCII Escape (Hex 1B) if not intercepted.

4. [Shift] [DEL ESC]

   This key sends an ASCII DEL (Hex 7F) if not intercepted.

5. [Back space]

   This key sends an ASCII Backspace (Hex 08) if not intercepted.

6. [Return]

   If not intercepted, this key sends an ASCII Carriage Return unless configured to send some other code.

NUMERIC/GRAPHICS PAD (10)

These keys operate in two modes: NUMERIC pad mode and GRAPHICS pad mode. [CTRL] [-] is used to toggle between the modes, and a indicator on line 27 shows when the mode is GRAPHICS. This toggle keycode can be intercepted by an application to lock the keypad in the NUMERIC mode in order to receive unique keycodes.

1. <u>NUMERIC mode</u> (no intercept)

   The keys return ASCII codes corresponding to the characters on the keycaps. [Shift] has no effect on the codes returned.

2. NUMERIC mode (with intercept)

   The keys return unique keycodes; that is, the Numeric-pad zero is different from the main keyboard zero, and all keys are independent of [Shift]. Note that the Numeric-pad [Tab] differs from the main keyboard [Tab] key in this regard. Applications can use these these keycodes to redefine this keypad.

3. GRAPHICS mode

   The active keys return the same keycodes (or else perform the same functions) as the corresponding graphics functions keys on the main keyboard (described previously). A template is used to label these functions.

   Since the graphics keycodes are not unique, applications should use AGIOS calls to force the pad to NUMERIC mode in order to use it.

NOTES ON KEYCODES

For information on the keycodes, refer to "Keycode Table" in Section 5.

1. [f1] through [f12] and the Numeric-pad keys (N-XXXX) return the same keycodes independently of [Shift] and [CTRL] operation. The qualifier byte can be examined to distinguish the various permutations.

2. The Numeric/Graphics pad keys only return unique keycodes when in NUMERIC mode. Applications can ensure proper operation by forcing NUMERIC and intercepting the [CNTL] [-] keys (code 70H plus [CNTL] qualifier bit).

3. Other functions not listed in the table are not intercept- able, namely:

   Soft Reset [Shift] [Reset Break]
   Hard Reset [CTRL] [Reset Break]
   Abort [CTRL] [Stop]
   User Key Menu [CTRL] [Menu]
   Touchscreen enable/disable [CTRL] [Shift] [Menu]
   Applications Keys [Shift] [User System]
   User Keys [CTRL] [User System]

4. If [CNTL] and [Shift] are typed together with a key which is not defined, the keycode returned is the same as if only [CTRL] were typed. The qualifier byte indicates both qualifiers, however.

LANGUAGES

WHAT TO DO ABOUT LANGUAGE

The HP 150 uses a superset of standard MS-DOS 2.1. All languages which are system independent and which don't require intensive screen usage *within the compiler* should work with no trouble.

The languages which have been used on the HP 150 and which should work include:

- BASIC:
  - BASIC/150 (MS BASIC-86)
  - MS BASCOM (BASIC-86 Compiler)
  - DRI CBII (Digital Research Compiler BASIC)
- Pascal:
  - MS Pascal
  - DRI Pascal MT/+ 86
  - Turbo Pascal
- C:
  - MS C (Lattice C)
  - DRI C
  - C-86
- Fortran:
  - MS Fortran-86
- Assembler:
  - MS MASM
  - DRI 8088/8086 Assembler

There are some languages which, as far as we know, have not been used in writing on the HP 150, but which should work with little difficulty. These are:

- Ryan-McFarland COBOL
- DRI Access manager
- DRI Screen Manager

For the above languages and utilities, you should be able to purchase the language locally either in HP disc format or in IBM-PC format (in which case you can trasnfer the language to the HP 150 using the utilities presented in Section 3).

Note that utilities provided with a language for use on a particular type of computer may not function at all on the HP 150.

One language is known to require special screen handling which is not readily converted. That language is MS COBOL. If you wish to use MS-COBOL, wait until it is available specifically for the HP 150.

And finally, as of this revision date, FORTH, UCSD p-System Pascal, and LOGO do not seem to be available on the HP 150.

Copyright Information