
BASIC Programmer's Library User's Guide

**by Miles Kehoe
David Pugmire
Brian Rainie**



*This manual has been written for
use with your HP Touchscreen
Personal Computer.*

**Manual Part No.
45310-90001**

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

Table of Contents

Chapter 1:

Getting Started with BPL

- 1-1** Introduction
 - 1-2** What Is BPL?
 - 1-3** Who Can Benefit From BPL?
 - 1-4** How Do I Use BPL?
 - 1-5** How Is BPL Organized?
 - 1-7** What Comes With BPL?
 - 1-8** Files On The BPL Disk
 - 1-11** A Sample Session With BPL
 - 1-11** Using BPL With The Sample Files
 - 1-12** Using BPL With Your Program
-

Chapter 2:

BPL Variables & Functions

Chapter 3:

Practical Programming

- 3-1** Introduction
- 3-2** A Philosophy of Programming the HP 150
- 3-3** Introduction to Escape Sequences
- 3-6** Summary: Using Escape Sequences
- 3-7** Introduction to Softkeys
- 3-8** User Definable Softkeys
- 3-10** Displaying the User Definable Softkeys
- 3-13** Programming Considerations
- 3-14** Controlling the Keyboard
- 3-16** How to get Transmit Function Key Mode Started
- 3-16** Programming Considerations
- 3-18** Controlling the Display
- 3-18** Display Enhancements
- 3-19** Character Set Selection
- 3-22** Alphanumeric Memory Control

3-22	Programming Considerations
3-23	Using Touch Screen
3-24	Defining a Touch Field
3-27	Types of Touch Fields
3-28	Row/Column Reporting
3-29	Specifying Reporting Modes
3-30	Deleting Touch Fields
3-31	Controlling User Defined Keys
3-31	Touch Screen Reset
3-32	Programming Considerations
3-35	Device Control from BASIC
3-36	Using AGIOS from BASIC

Chapter 4:

Variables and Functions by Group

4-1	AGIOS
4-1	Alpha
4-1	Display Control
4-2	Editing
4-2	Softkey Control
4-2	General
4-2	Graphics
4-2	Display Control
4-3	Origins
4-3	Polygons
4-3	Text
4-3	Vectors
4-4	Internal
4-4	Strings
4-4	Numeric
4-5	Touch Screen
4-5	Subroutines
4-5	String
4-5	Numeric

Chapter 5:

Alphabetic Listing of Variables & Functions

1

Getting Started with BPL

Introduction

The BASIC Programmer's Library, 'BPL' for short, is a collection of subroutines, functions and variable definitions. By utilizing the **BPL** whenever you write a program in BASIC, you can save a great deal of time and effort. **BPL** helps make you more productive by saving you the effort of re-inventing the wheel every time you start a project in BASIC.

In this chapter, we will answer the following questions about **BPL**:

- What Is **BPL**?
- Who Can Benefit From **BPL**?
- How Do I Use **BPL**?
- How is **BPL** Organized
- What Comes with **BPL**?
- A Sample Session With **BPL**

BPL assumes you are somewhat familiar with BASIC. If you feel you need to review the language, refer to the owner's manual for your version of BASIC. There are also a number very good books useful in learning or reviewing the language. These books are generally available from a local computer dealer or bookstore.

The following manuals are referred to in this manual:

<i>HP150 Owner's Guide</i>	45621-90001
<i>HP150 Terminal User's Guide</i>	45623-90001
<i>Using Your HP Touchscreen Personal Computer</i>	45626-90004
<i>Using You HP Touchscreen Personal Computer as a Terminal</i>	45626-90005
<i>Series 100/BASIC User's Guide</i>	45445-90001
<i>HP System Programmer's Pack</i>	45435A

Let's get started!

What Is BPL?

BPL is a collection of subroutines, variable definitions, and function definitions. By using **BPL**, you have access through your program to virtually all of the features of the HP Touchscreen PC including:

- Screen control functions
- Keyboard Control
- Full Use of the Touchscreen
- Control of the User-defined Function Keys
- Line and Math Character Sets
- Access to BIOS Routines via AGIOS Calls
- Access to Datacomm and Plotter Devices

The subroutines are utility subprograms which give you a consistent method of performing console input, displaying error messages, and making system calls.

The entire philosophy of **BPL** is to make you productive in BASIC right away. You don't need to spend hours learning the HP Touchscreen PC. You don't need to develop 'standards' for programs written in BASIC. **BPL** includes a structured organization and fixed definitions so you can get off to a fast start with your HP Touchscreen PC. This is true for both the *Series 100 BASIC Interpreter* and the *Series 100 BASIC Compiler*.

BPL does not attempt to explain the various escape sequences, and system (AGIOS) calls in depth. Rather, the most useful of the sequences and calls have been incorporated into **BPL**. You will then see *how* to use these 'most common' features in your program in such a way that you will be productive almost immediately.

BPL *does* explain how you can incorporate your own functions and definitions so the overall effect will be a well-organized final product for you. But if you want additional information on the various functions, refer to the appropriate system manuals available from Hewlett-Packard.

Who Can Benefit From BPL?

The overall goal of **BPL** is to simplify your life as a programmer. The examples are provided specifically for someone programming in *Series 100 BASIC*. Both the compiler and the interpreter are supported. Whether you are programming for your own purposes, or programming a package for use by someone else, **BPL** will improve your productivity.

However, the principles illustrated apply to programming the HP Touchscreen PC from any language. For example, if you are programming in Pascal you will find the functions and variable definitions easily converted. In fact, the same general principles apply to programming the HP Touchscreen PC *terminal* regardless of the type of 'host' you use. This is true whether your host system is an HP 3000 or a minicomputer from another vendor. If you are writing an application which uses the HP Touchscreen PC as nothing more than a terminal, the principles presented here will help you.

In short, if you are programming any application that uses the HP Touchscreen PC, you can probably benefit from owning **BPL**.

How Do I Use BPL?

BPL is provided on disc as a BASIC program. By loading **BPL** into memory when you first begin to write a program, you can immediately start *YOUR* program. This saves you the time of entering all the standard escape sequences you will use in *EVERY* program, making you more productive.

BPL, like any programming aid, expects a few things of you. First, **BPL** expects your 'main program' to start at line 5000. The lines before 5000 are, for the most part, reserved for **BPL**. More will be said about this later in the section titled '**How Is BPL Organized?**'.

Starting your application at line 5000 is not as severe a restriction as it may seem. *Series 100 BASIC* allows line numbers to range as high as 65529, so you will likely run out of memory before you run out of line numbers!

It might seem 'more efficient' to renumber **BPL** and 'compact' the source code to use fewer line numbers. In fact, doing so does *not* increase the available memory space and *does* assure a lot of extra work on your part should enhancements to **BPL** become available!

BPL also expects certain variables to be 'reserved'. While this is not a limit on you, be sure to check the 'reserved variables' listed in Chapter 5. You should not change the definition of these reserved variables or **BPL** may not function properly. If strange things begin to happen, check to see if you've redefined a reserved variable or function!

You will start your programming session by loading **BPL**. Be sure, when you save your program, not to name it '**BPL**!' If this does happen, you will want to use another copy of **BPL** for other programs you write. There is a section later in this chapter titled '**A Sample Session with BPL**' which illustrates how you would use **BPL** in a simple application.

Once **BPL** is loaded, proceed to write your program starting at line 5000. Remember, lines below 5000 are generally reserved for special purposes by **BPL**. Refer to the section below titled '**How is BPL Organized**' for a detailed map of available line ranges.

Also, remember that all ranges not marked as 'Available for User Definitions' should not be used. These are reserved for enhancements and fixes in any future releases of **BPL**.

How is BPL Organized?

BPL is organized in a specific way so that variables, functions, and subroutines are located within different ranges of line numbers. Those variables and functions which control similar functional components of the HP150 are 'grouped' in the same general range of line numbers. Within each of these groupings, variables are defined first and functions are defined second.

Furthermore, within the entire library, those lines which define escape sequence control variables and functions consistently end in zero. That is, all lines which are multiples of 10 contain variable and function definitions which involve escape sequences. Those lines which end in 5 define AGIOS variables and functions.

In summary, those lines with line numbers which end with a '5' are AGIOS related definitions. Those lines with line numbers ending in '0' are escape sequence related definitions. This convention applies only within the Library, that is, lines numbered below 5000.

Let's take a closer look at the ranges reserved for the several general functions.

Line Range	Functional Area
1000 - 1030	Program identification for your application
1040	Set up information (Interpretive BASIC ONLY)
1050	Setup for console and printer width
1060	BPL Revision and Date Code
1100 - 1135	BPL Identification and Copyright Notice
1140 - 1199	Available for User (Identification and Copyright)
1200 - 1299	General Variable and Function Definition
1300 - 1499	Available for User (General) Definitions
1500 - 1799	Alphanumeric Control Variables and Functions
1800 - 1999	Available for User (Alphanumeric) Definitions
2000 - 2399	Graphics Control Variables and Functions
2400 - 2499	Available for User (Graphics) Definitions
2500 - 2599	Touchscreen Control Variables and Functions
2600 - 2699	Available for User (Touchscreen) Definitions
2700 - 2945	AGIOS Specific Variables and Functions
2955 - 2995	Available for User (AGIOS) Definitions
2999	GOTO Start of Main Program - REQUIRED! If this line is removed, subroutines will be executed before your main program. Most likely the error reported will be RETURN WITHOUT GOSUB AT LINE 3040 after you press any key.
3000 - 3040	Subroutine CHRIN (CHaRacter INput) to accept a single character from the console. If file 'BPL3000A.BAS' was MERGE d , input will be accomplished via an assembly language call. This requires use of AGIOS subroutines. Use the appropriate file whether you are using the interpreter or compiler.
3050 - 3095	Subroutine LININ (LINE INput) to accept a line of text into the variable BUF\$. LININ uses the CHRIN. Found in file BPL3050.BAS.
3100 - 3110	Subroutine ECHO to display the character provided by CHRIN to the display at the current cursor location. Found in file BPL3100.BAS.

1-6 Getting Started with BPL

3150 - 3160

Subroutine ERROR to provide an orderly halt should an error occur. To use this subroutine with the compiler, be sure to specify the appropriate compiler option. Refer to your compiler manual for additional information. Note that line 3160 is a STOP instruction. You may wish to modify this routine for error recovery in your application. Found in file BPL3150.BAS.

3500 - 4999

User-defined subroutines.

5000 - 65529

Main Program. Your application 'main' code should be within this range of lines.

What Comes with BPL?

BPL is composed of a disc and this manual. Using this manual, you can learn what is available as part of BPL and learn how to use the various components of BPL.

The disc contains several files. Some are the source code to BPL. These are provided in a variety of forms which allows you to customize BPL for your application. For example, the parts of the library which deal with AGIOS system calls are contained in one file. If you wish to use those calls in your application, you can MERGE those lines into your code. If you don't wish to use those calls, you can simply load the 'non-AGIOS' BPL. The disc also contains several sample programs which you will find in this manual. We encourage you to look at these sample programs: each is provided to illustrate a point made in this manual.

Finally, there are several files that allow you to perform input and output functions with many of the HP Touchscreen PC devices such as AUX, CON, LST, PRN, INT, PLT, COM, COM1, and COM2. These files are independent of the regular BPL files and require renumbering to be used with BPL files so that their lines will fall in the proper range of BPL line numbers.



Files on the BPL Disc.

The disc which contains **BPL** contains the following files:

File Name	Description
INDEX.DOC	Complete index for this manual. Print using either your word processor or the MS-DOS COPY command.
BPL.BAS	The full BPL containing both standard and AGIOS functions and subroutines.
BPLS.BAS	BPL using only standard calls: no AGIOS calls are included.
BPLAI.BAS	The AGIOS definitions of BPL for use with interpreted BASIC only.
BPLAC.BAS	The AGIOS definitions of BPL for use with compiled BASIC only.
BPL3000A.BAS	Subroutine for single byte input from the console using the AGIOSI.TLA (assembly language) routine. The character is accepted <i>without</i> echo.
BPL3000B.BAS	Subroutine for single byte input from the console using BASIC only. The character is accepted <i>without</i> echo.
BPL3050.BAS	Subroutine for accepting a full line of data from the console. This uses whichever routine has been loaded at line 3000.
BPL3100.BAS	Subroutine for echoing the character from line 3000 to the console. This routine is used regardless of the character input routine used (B3000A.BAS or B3000B.BAS).
BPL3150.BAS	Subroutine to display a BASIC error message at the left margin of line 20 of the display. This routine can be replaced or modified to suit the needs of your application.
AGIOSI.ASM	The source code of AGIOSI.TLA. Note that the assembler does NOT directly generate the code in AGIOSI.TLA! If you re-assemble this file, you will destroy your specially processed copy of AGIOSI.TLA!
AGIOSI.IMG	The assembly language routines used by the interpretive version of BPL for access to AGIOS. This file has been specially processed <i>AFTER</i> assembly as mentioned above. Refer to the <i>Series 100 BASIC Manual</i> for additional details.

1-8 Getting Started with BPL

AGIOSC.ASM	The source code of AGIOSC.OBJ. Unlike the interpretive version of this file, the assembler generates this code directly. This file should be used when linking your BASIC compiled program.
AGIOSC.OBJ	The assembly language routines used by the compiled version of BPL . When you compile your application, no special BPL files must be on-line during execution.
AGIOS.DOC	Documentation file for AGIOS calls. Print using either your word processor or the MS-DOS COPY command.
SAMPLE1.BAS	Sample file which illustrates the use of alternate character sets including the line drawing and math character sets.
SAMPLE2.BAS	Sample file illustrating the use of the limited tone generation facilities on the system.
SAMPLE3.BAS	Sample file showing how to use area pattern definition.
SAMPLE4.BAS	Sample file illustrating the use of character enhancements such as inverse video and blinking text.
SAMPLE5.BAS	Sample file showing how to use graphics text in any of the defined sizes.
SAMPLE6.BAS	Sample file showing how to set the orientation of graphics text, such as inverted text.
SAMPLE7.BAS	Sample file illustrating point plotting.
SAMPLE8.BAS	Sample file which illustrates user-defined area pattern definition.
SAMPLE9.BAS	Sample file illustrating the use of the various line patterns available.
SAMPLE10.BAS	Sample file which shows how to draw lines in graphics mode.
SAMPLE11.BAS	Sample file showing the use of 'memory lock'.
SAMPLE12.BAS	Sample file illustrating the use of 'rubber band lines'.
SAMPLE13.BAS	Sample file showing use of touch screen.
SAMPLE14.BAS	Sample file which uses polygonal area fill and boundry pen.
SAMPLE15.BAS	Sample file showing how to use origins on the system.
SAMPLE16.BAS	Sample file illustrating the set, complement, jam, and clear modes of graphics memory.
SAMPLE17.BAS	Sample file showing how to define custom patterns for drawing lines.

SAMPLE18.BAS	Sample file which illustrates the User-definable area fill patterns.
SAMPLE19.BAS	Sample file illustrating the use of user-defined softkeys.
DFUNCSIB.ASM	Assembler source code for DOPEN, DACCESS, and DCLOSE routines. These routines are used to access HP Touchscreen PC I/O devices AUX, CON, LST, PRN, INT, PLT, COM, COM1, and COM2. For use with intrepreative BASIC.
DFUNCSCB.ASM	Compiled BASIC version of DFUNCSIB.ASM.
DFUNCSCB.OBJ	Object code of FUNCSCB.ASM to be used with compiled BASIC programs.
DFUNCSIB.OBJ	Object code of FUNCSIB.ASM to be used with interpretive BASIC programs.
TESTIB.BAS	Interpretive BASIC program showing how to use I/O device routines.
TESTCB.BAS	Source code for compiled BASIC program showing how to use the I/O device routines.
TESTCB.OBJ	Object code of TESTCB.BAS that may be linked with your compiled BASIC program.
TESTCB.EXE	Executable version of TESTCB.BAS
DFUNCSIB.IMG	Memory image of assembly language routines in BLOAD format. Loaded by your intrepreative BASIC program to perform the I/O functions provided. This file is used by example program TESTIB.BAS.
DATEST.BAS	Example of using these device I/O routines to interact with a host computer.
PLTEST.BAS	Example of using these device I/O routines to interact with a plotter.
DEVICE.DOC	Documentation file for device I/O routines. Print using either your word processor or the MS-DOS COPY command.

A Sample Session with BPL

Now that you are aware of what **BPL** is an overview of an actual session using **BPL** may aid you in your initial uses of the BASIC Programming Library. First we will show an example of using the sample files with **BPL** and then we will write a simple program using some of the **BPL** functions.

Note that **BPL** is not installable under P.A.M. as a programming tool. **BPL** is intended to be used with BASIC. Your application written, using **BPL**, can be installed under P.A.M. by referring to the HP 150 User's Guide.

Using BPL with the Sample Files

After having loaded BASIC type:

```
LOAD "BPL" 
```

This will load the standard **BPL** without any subroutines or AGIOS. Next merge in a sample file by typing in the file name you want, i.e.:

```
MERGE "SAMPLE4" 
```

Then to see how the sample illustration works run the program by typing:

```
RUN 
```

You should see the phrase "GOOD TRY MY FRIEND" with some different character enhancements. If you want to try another sample program you need to delete lines 5000 to the end of the program. For **SAMPLE4** you would type:

```
DELETE 5000-5050 
```

You may now merge in another sample file and run it as you did above.

Using BPL with Your Program

In this session we will load the standard **BPL** without subroutines or AGIOS and start writing a custom program that uses a few of the **BPL** features. To begin, after having entered BASIC, load the standard **BPL** by typing:

```
LOAD "BPL" 
```

Use the BASIC edit command or the HP Touchscreen PC **LINE MODIFY** function to alter lines 1000 to 1030 to reflect the proper program identification, copyright, etc. for this program. Now begin writing your program at line 5000. For example you might type:

```
5000 REM Start of Main Program
5010 REM This Program will type out TEST 10 times
5020 REM
5030 PRINT CLS$ "CLEAR THE SCREEN"
5040 FOR I = 1 TO 10
5050 PRINT FNLOCATE$(I,I*4); "TEST"
5060 NEXT
5070 END
```

After having run the program be sure to save it with a new name other than BPLS so as not to overwrite your original BPLS file. Save this program as TEMP by typing:

```
SAVE "TEMP.BAS",A 
```

You are now ready to enjoy the benefits of using **BPL** to improve your BASIC programming productivity. The sections that follow are primarily for reference. Chapter 2 is a reference to every **BPL** variable, Chapter 3 is filled with some practical programming tips for the HP Touchscreen PC, and Chapter 4 contains some brief notes on advanced features of the HP Touchscreen PC and using the datacom files found on the **BPL** disk.

2

BPL Variables and Functions

ADOFF\$

Statement:

ADOFF\$=ESC\$+“*dF”

Type:

Alpha, Display

Purpose:

To turn off the alpha display.

Remarks:

ADOFF\$ turns off the alphanumeric display. The data in the two pages of alphanumeric screen memory is not affected.

Example:

5000 PRINT ADOFF\$

ADON\$

Statement:

ADON\$=ESC\$+“*dE”

Type:

Alpha, Display

Purpose:

To turn on the alpha display.

Remarks:

ADON\$ turns on the alphanumeric display and alpha numeric cursor. The data in alphanumeric memory is not affected.

Example:

5000 PRINT ADON\$

ALTCHROFF\$

Statement: ALTCHROFF\$ = CHR\$(15)

Type: Alpha, Display

Purpose: To end the use of the alternate character set.

Remarks: Turns off the use of the alternate character set and returns you to the normal characters. This is equivalent to typing **CTRL O** from the keyboard. A carriage return will also end the use of the alternate character set.

Example: 5000 Print ALTCHROFF\$
(See ALTCHRON\$ for a further example.)

ALTCHRON\$

Statement: ALTCHRON\$ = CHR\$(14)

Type: Alpha, Display

Purpose: To start the use of the alternate character set.

Remarks: Starts the use of the alternate character set which has been selected using FNALTCHR\$. This is equivalent to typing **CTRL N** from the keyboard.

It is important to follow the use of ALTCHRON\$ by a semicolon **;** when PRINT'ed to avoid an unwanted carriage return.

Example: SAMPLE1.BAS (On **BPL** disc)

```
5000 PRINT FNALTCHR$ ("B")           'Select line drawing set
5010 PRINT CLS$                     'Clear Screen
5020 PRINT FNLOCATE$ (4,20);         '5030-5070 draws a box
5030 PRINT ALTCHRON$;"R,,,,,,,,,T"
5040 PRINT FNLOCATE$ (5,20);
5050 PRINT ALTCHRON$;"          ."
5060 PRINT FNLOCATE$ (6,20);
5070 PRINT ALTCHRON$;"F,,,,,,,,,G"
5080 PRINT ALTCHROFF$
```

2-2 BPL Variables and Functions

BDRYPENOFF\$

Statement:

`BDRYPENOFF$ = ESC$+""*mH"`

Type:

Graphics, Polygon

Purpose:

To turn off the drawing of a boundary around a polygon. This is the default state.

Example:

```
5000 PRINT BDRYPENOFF$  
(see BDRYPENON$ for a further example)
```

BDRYPENON\$

Statement:	BDRYPENON\$ = ESC\$+""*m1H"
Type:	Graphics, Polygon
Purpose:	To draw a solid boundary around a polygon.
Remarks:	<p>This variable selects the pen to be used to draw the boundary of a filled polygon. Because this is a monochrome system the actual value used is not significant. Here, the digit one '1' was used.</p> <p>This variable turns on the ability to draw a solid line around the polygon.</p> <p>There is the ability to control the drawing of a boundary on individual sides of a polygon. The character sequence ""*pu" will lift the boundary pen and ""*pv" will lower it. Once one of these conditions is selected, all subsequent edges of the polygon will be drawn with the same condition until a change is made. These switches are used by placing them at the desired location in the polygon vector list. (""*pu" & ""*pv" must be lowercase letters or the escape sequence will be terminated.)</p>
Example:	5000 PRINT BDRYPENON\$
Example:	SAMPLE14.BAS (on BPL disc)

```
5000 PRINT GSETUP$           'Prepare to use graphics
5010 PRINT FNMOVE$ (100,100) 'Move graphics pen to starting point
5020 PRINT FNFILLPAT$ (6)    'Select non-solid fill pattern
5030 PRINT BDRYPENON$       'Turn on boundary pen
5040                         'Draw polygon raising & lowering boundry pen
5050 PRINT POLYGON$+ ""100,100,100,200,*pu,200,200,*pv,200,100,*pu,100,100T"
5060 PRINT GPDFTS;FNMOVE$ (50,50) 'Clean exit from sample
5070 PRINT FNGTLABEL$ ("Press Return to exit.")
5080 INPUT A$                'Accept key input to return to alpha
5090 PRINT GCLSS$,ADONS$     'Clear graphics & turn on alpha
```

2-4 BPL Variables and Functions

BELL\$

Statement:	BELL\$ = CHR\$(7)
Type:	General
Purpose:	To ring the bell.
Remarks:	Further tones can be made by making use of the BASIC "OUT" command (See SAMPLE2.BAS below).
Example:	5000 PRINT BELL\$ SAMPLE2.BAS (On BPL disc)



```
5000 P1= & H19           'Tones are at port & H19
5010 GOSUB 5070
5020 A$=INPUT$(1):IF A$<"0" OR
    A$>"9" THEN 5020     'Tones may be 48 to 64, decimal
5030 A%=VAL(A$)          ' example uses '0' to '9'.
5040 OUT P1,ASC(A$)      'Generate tone
5050 PRINT A%           'Echo tone number
5060 GOTO 5020
5070 PRINT CLS$
5080 PRINT FNLOCATE$(2,10);
5090 PRINT "Sample tones will be generated by pressing keys 0 - 9."
5100 PRINT FNLOCATE$(3,10);"Type "CTRL" and "C" at the same time to exit."
5110 RETURN
```



The above makes use of an HP Touchscreen PC specific port address. Using this in your programs may prevent your program from running on future 150 compatible computers.

BSS

Statement: BS\$ = CHR\$(8)

Type: Alpha, Edit

Purpose: To move the alpha cursor back one space.

Remarks: This is the same as pressing Back space .

You must follow BS\$ with a semicolon to prevent a carriage return from being appended when PRINT'ed.

Example: 5000 PRINT BS\$;

BTAB\$

Statement: BTAB\$ = ESC\$+"i"

Type: Alpha, Edit

Purpose: To move the alpha cursor back to the previous tab stop.

Remarks: Functions the same as pressing back tab (Shift Tab) on the keyboard.

You must follow BTAB\$ with a semicolon to prevent a carriage return from being appended when PRINT'ed.

Example: 5000 PRINT BTAB\$;

CLL\$

Statement: CLL\$ = ESC\$+"K"

Type: Alpha, Edit

Purpose: To clear from the cursor to the end of the line.

Remarks: This variable functions the same as Clear line clearing the line from the cursor to the end of the line.

Example: 5000 PRINT CLL\$;

2-6 BPL Variables and Functions

CLRS

Statement: `CLR$ = ESC$+"J"`

Type: Alpha, Display

Purpose: To clear the alpha screen from cursor to end of screen.

Remarks: Works the same as pressing `Clear display` .

Example: `5000 PRINT CLR$;`

CLS\$

Statement: `CLS$ = HOMES$ + CLR$`

Type: Alpha, Display

Purpose: To clear the entire display.

Remarks: This variable works the same as pressing `Shift` `Clear display` or pressing `Clear display` followed by `Clear display` .

This is also an example of how two variables may be combined to create a third.

Example: `5000 PRINT CLS$`

COFF\$

Statement: `COFF$ = ESC$+"*dR"`

Type: Alpha, Display

Purpose: To turn off the alpha cursor.

Remarks: It is often desirable to turn off the alpha cursor so that the user of a program does not think a response is required.

Example: `5000 PRINT COFF$`

CON\$

Statement: `CON$ = ESC$+"*dQ"`

Type: Alpha, Display

Purpose: To turn on the alpha cursor.

Remarks: Alpha cursor on is the default state.

Example: `5000 PRINT CON$;`

CR\$

Statement: `CR$ = CHR$(13)`

Type: Alpha, Edit

Purpose: To issue a carriage return.

Remarks: Functions the same as pressing from the keyboard.

You must follow this with a semicolon or BASIC will add an additional carriage return when PRINT'ed.

Example: `5000 PRINT CR$;`

DL\$

Statement: `DL$ = ESC$+"M"`

Type: Alpha, Edit

Purpose: To delete the line the cursor is on.

Remarks: Functions the same as pressing .

Example: `5000 PRINT DL$`

ENHOFF\$

Statement: ENHOFF\$ = ESC\$+"&d@"

Type: Alpha, Display

Purpose: To end the use of the selected character enhancement.

Remarks: ENHOFF\$ will stop the use of the current enhancement. A carriage return will also stop the use of an enhancement.

Example: 5000 PRINT ENHOFF\$
(see FNENHON\$ for a further example)

ESCS

Statement: ESC\$ = CHR\$(27)

Type: General

Purpose: To represent the escape character.

Remarks: ESC\$ is defined to make it easier to use escape sequences to control the computer.

Example: 5000 PRINT ESC\$;

FNALTCHR\$(BUF\$)

Statement: DEF FNALTCHR\$(BUF\$) = ESC\$+"'" +BUF\$

Type: Alpha, Display

Purpose: To select the alternate character set.

Remarks: Two alternate character sets are available for use from BASIC, Line Drawing and Math. To use Bold Face Roman and Italic Roman, you must use AGIOS area operations.

The code characters passed to the function are:

- @ Normal Roman
- A Math
- B Line Drawing

Example: 5000 PRINT FNALTCHR\$ ("B")
(See ALTCHRON\$ for a further example.)

FNBOX\$(X1,Y1,X2,Y2)

Statement: DEF FNBOX\$(X1,Y1,X2,Y2) = FNMOVE\$(X1,Y1) +
FNDRAW\$(X2,Y1) +
FNDRAW\$(X2,Y2) +
FNDRAW\$(X1,Y2) +
FNDRAW\$(X1,Y1)

Type: Graphics, Vector

Purpose: To draw a rectangle by giving the opposite corners.

Remarks: This function uses other functions to perform the task of drawing a rectangle. First the pen is moved to the beginning corner, then each side of the box is drawn.

Example: 5000 PRINT GDON\$;
5010 PRINT FNBOX\$ (150,100,300,250);

2-10 BPL Variables and Functions

FNBOXFA\$(X1,Y1,X2,Y2)

Statement:

```
DEF FNBOXFA$(X1,Y1,X2,Y2) = ESC$+"*m"+STR$(X1)
                             +","+STR$(Y1)+","+STR$(X2)
                             +","+STR$(Y2)+"E"
```

Type:

Graphics, Polygon

Purpose:

To place a shaded rectangle on the graphics screen.

Remarks:

This function uses the currently defined fill pattern (see FNFILLPAT\$) to shade the designated area of the graphics screen. The user defines rectangular region by specifying the lower left and upper right corners. The corners are absolute coordinates with the lower left corner of the screen being (90,0). The box does not have a boundary.

Coordinates must be specified by lower left corner, upper right corner.

Example:

```
5000 PRINT GDON$;
5010 PRINT FNBOXFA$ (125,150,325,350);
```


FNCENTER\$(BUF\$,W)

Statement:

```
DEF FNCENTER$(BUF$,W) = STRING$(INT((W-LEN
(BUF$))/2), " ") + BUF$
```

Type:

Alpha, Display

Purpose:

To center a phrase on a line.

Remarks:

This function will center a phrase on a line. The phrase to be centered is in the variable BUF\$ and the length of the line is W.

Example:

```
5000 PRINT FNCENTER$ ("Good Try",80)
```

FNDEFLINE\$(X1,X2)

Statement:

```
DEF FNDEFLINE$(X1,X2) = ESC$ + "*"m" + STR$(X1) + "," +
STR$(X2) + "C"
```

Type:

Graphics, Vector

Purpose:

To define the User-definable line pattern.

Remarks:

The first parameter is a number between 0 and 255. The system will use the binary representation of that number to define the eight bits (screen dots) to make up the pattern. The second parameter is a scaling factor of from 1 to 16. This factor determines the number of times each dot in a pattern is repeated before starting the next dot.

Example:

SAMPLE17.BAS (on BPL disk)

```
5000 PRINT GRESET$,GSETUP$           'Prepare to use graphics
5010 PRINT FNDEFLINE$ (170,3)         'Define a line pattern & scale
5020 PRINT FNLINEPAT$ (2)             'Select user defined line pattern
5030 PRINT FNDRAW$ (300,300)          'Draw a sample line
5040 PRINT GPDFT$,FNMOVES$ (80,50);  'clean exit from sample
5050 PRINT FNGTLABEL$ ("Press Return to exit.");
5060 INPUT AS$
5070 PRINT GCLSS$,ADON$
```

FNDEFPAT\$(BUF\$)

Statement:	DEF FNDEFPAT\$(BUF\$)=ESC\$+"*m"+BUF\$+"D"
Type:	Graphics, Polygon
Purpose:	To define the user-defined area fill pattern.
Remarks:	With this function the user can define any type of area fill pattern desired and then use area fill commands to use that pattern. The string required by this function is a list of eight numbers separated by commas and enclosed in double quote marks. Each number must be between 0 and 255. The system will use the binary representation of that number to make up the fill pattern. The first number in the string is for the bottom row of eight bits in an eight by eight cell, the last number for the top row.
Example:	SAMPLE18.BAS (on BPL disc)

```
5000 PRINT GRESET$,GSETUP$           'Prepare to use graphics
5010                                'Define area pattern
5020 PRINT FNDEFPAT$ ("51,51,204,204,51,51,204,204")
5030 PRINT FNILLPAT$ (2)             'Select user defined pattern
5040 PRINT FNBOXFA$ (100,100,200,200) 'Box to illustrate pattern
5050 PRINT GPDFTS$;FNMOVES$ (50,50)  'Clean exit from sample
5060 PRINT FNGTLABEL$ ("Press Return to exit.")
5070 INPUT A$
5080 PRINT GCLS$,ADONS$
```

FNDRAW\$(X1,Y1)

Statement:

```
DEF FNDRAW$(X1,Y1) = ESC$+"*pb"+STR$(X1)+","+  
STR$(Y1)+"Z"
```

Type:

Graphics, Vector

Purpose:

To draw a line from the current pen position.

Remarks:

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

Example:

```
5000 PRINT FNDRAW$ (511,389);
```



FNDRAWMODE\$(X1)

Statement: DEF FNDRAWMODE\$(X1) = ESC\$+ "*"m"+STR\$(X1)+"A"

Type: Graphics, Display

Purpose: To select the vector drawing mode.

Remarks: The vector drawing modes are as follows:

- 0 Graphics Memory Not Changed
- 1 Clear Mode
- 2 Set Mode
- 3 Complement Mode
- 4 Jam Mode

The data in graphics memory is either 0 (off) or 1 (on). Drawing modes determine what effect a graphics command (Vector or Polygon) will have on graphics memory.

CLEAR MODE will turn the selected display bits in graphics memory to 0 or *OFF*. The selected bits are those that are *ON* in the line or area pattern. Clear mode has the effect of drawing dark lines on a light background. If the background is dark there is no noticeable effect on the display.

SET MODE is the opposite of clear mode giving light lines on a dark background. As in clear mode only the bits that are *ON* in the pattern are affected.

COMPLEMENT MODE causes the selected bits (those that are *ON*) in a pattern to toggle (on to off, off to on). The pattern drawn is the opposite of whatever the graphics status is at that point. Complement mode gives you the ability to selectively erase. If a line is drawn and then another line is drawn over the top of the first, the first appears to have been erased.

JAM MODE differs from the other 3 modes in that all the bits of a pattern affect the display. Jam mode gives the effect of having cut out the pattern and placed it over the top of the display. The example below shows how the jam mode pattern appears with graphics memory set to *OFF* and then with it set to *ON*.

Example:

SAMPLE16.BAS (on BPL disc)

```
5000 PRINT GRESET$,GSETUP$           'Prepare to use graphics
5010 PRINT FNILLPAT$ (6)             'Set fill pattern
5020 PRINT FNDRAWMODE$ (1)          'Set Clear Mode
5030 PRINT FNBOXFA$ (0,0,100,100)
5040 PRINT FNDRAWMODE$ (2)          'Set Set Mode
5050 PRINT FNBOXFA$ (100,100,200,200)
5060 PRINT FNDRAWMODE$ (3)          'Set Complement Mode
5070 PRINT FNBOXFA$ (200,200,300,300)
5080 PRINT FNDRAWMODE$ (4)          'Set Jam Mode
5090 PRINT FNBOXFA$ (300,300,400,400)
5100 PRINT FNDRAWMODE$ (3)
5110 PRINT FNMOVE$ (110,50);FNGTLABEL$ ("Clear Mode")
5120 PRINT FNMOVE$ (210,150);FNGTLABEL$ ("Set Mode")
5130 PRINT FNMOVE$ (310,250);FNGTLABEL$ ("Complement Mode")
5140 PRINT FNMOVE$ (410,350);FNGTLABEL$ ("Jam Mode")
5150 PRINT FNMOVE$ (0,300)
5160 IF FLAG=0 THEN 5180
5170 PRINT FNGTLABEL$ ("Press Return to End");GOTO 5190
5180 PRINT FNGTLABEL$ ("Press Return to Continue")
5190 INPUT A$
5200 IF FLAG=0 THEN 5220
5210 PRINT GCL$,ADON$:END
5220 PRINT GSET$                     'Set graphics to all on
5230 FLAG=1:GOTO 5020
```

FNENHON\$(BUF\$)

Statement: DEF FNENHON\$(BUF\$) = ESC\$+"&d"+BUF\$

Type: Alpha, Display

Purpose: To define the type of enhancement to be made to the characters that follow.

Remarks: Enhancements are defined according to the following table:

enhancement:	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
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		x

FNENHON\$ must be followed by a semicolon (;) to prevent the automatic insertion of carriage return.

Example: SAMPLE4.BAS (on **BPL** disc)

```
5000 PRINT CLS$           'Clear Screen
5010 PRINT FNENHON$( "C"); 'Inverse blinking enhancement
5020 PRINT "GOOD TRY";
5030 PRINT ENHOFF$;
5040 PRINT FNENHON$( "D"); 'Underline enhancement
5050 PRINT " MY FRIEND."
```

FNfillsPAT\$(X1)

Statement:	DEF FNfillsPAT\$(X1) = ESC\$ + "*"m" + STR\$(X1) + "G"
Type:	Graphics, Polygon
Purpose:	To select the desired fill pattern.
Remarks:	This function selects a pattern for polygonal and rectangular area fill. The possible patterns are: <ol style="list-style-type: none">1 Solid fill pattern (Default)2 User-defined pattern (see FNDEFPAT\$)3-10 Predefined fill patterns.
Example:	SAMPLE8.BAS (on BPL disc)

```
5000 PRINT GSETUP$           'Prepare to use graphics
5010 FOR I = 3 TO 10
5020 PRINT FNfillsPAT$(I)    'Set fill pattern
5030 R= (I-2) * 50
5040 PRINT FNBOXFA$(R,200,R+45,300) 'Show fill patterns 3 to 10, solid not
5050 NEXT I                  ' shown
5060 PRINT GPDFT$;FNMOVES$(50,50) 'clean exit from sample
5070 PRINT FNGTLABEL$ ("Press Return to exit.")
5080 INPUT A$
5090 PRINT GCLS$;ADONS$
```

FNGCMA\$(X1,Y1)

Statement:

```
DEF FNGCMA$(X1,Y1) = ESC$+"*d"+STR$(X1)+","+  
STR$(Y1) + "O"
```

Type:

Graphics, Display

Purpose:

To move the graphics cursor to specified coordinates.

Remarks:

This graphics cursor move is in relation to the absolute lower left of the screen, coordinates (0,0). The cursor move takes place even if the cursor is off. The upper right corner is (511,389).

Example:

```
5000 PRINT FNGCMA$ (250,300);
```

FNGCMR\$(X1,Y1)

Statement:

```
DEF FNGCMR$(X1,Y1) = ESC$+"*d"+STR$(X1)+","+  
STR$(Y1) + "P"
```

Type:

Graphics, Display

Purpose:

To move the graphics cursor.

Remarks:

This function moves the graphics cursor relative to the current position of the cursor.

Example:

```
5000 PRINT FNGCMR (25,-25);
```

FNGTLABEL\$(BUF\$)

Statement:

DEF FNGTLABEL\$(BUF\$) = ESC\$+"*1"+BUF\$

Type:

Graphics, Text

Purpose:

To place graphics text on the graphics screen.

Remarks:

This is the function used to write any graphics labels or text on a graphics screen. It can be combined with the size and rotate command for further control. Default text size is one.

Example:

SAMPLE5.BAS (on BPL disc)

```
5000 PRINT GSETUP$           'Prepare Graphics for use
5010 FOR I=1 TO 8
5020 PRINT FNGTSIZE$(I)      'Set graphics text size
5030 PRINT FNMOVE$(50,40*I)  'Move pen to new position for each label
5040 PRINT FNGTLABEL$("GTEXT") 'Print Graphics label
5050 NEXT I
5060 PRINT FNMOVE$(10,30);FNGTSIZE$(1) 'Clean exit from sample
5070 PRINT FNGTLABEL$("Press Return to exit.")
5080 INPUT A$
5090 PRINT GCLS$,ADONS
```

FNGTORGN\$(X1)

Statement:

```
DEF FNGTORGN$(X1) = ESC$ + "*"m" + STR$(X1) + "Q"
```

Type:

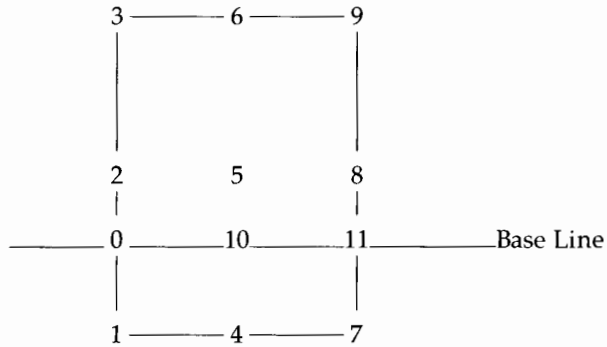
Graphics, Text

Purpose:

To set the justification of a graphics text character.

Remarks:

This function sets the graphics text origin to one of twelve positions of text justification. The possible justification positions are outlined in the following representation of a graphics text cell. Position 1 is the default.



If a position other than one on the left is chosen the text will not appear on the screen until **Return** is sent.

Example:

```
5000 PRINT FNGTORGN$(8);
```

FNGTROT\$(X1)

Statement:	DEF FNGTROT\$(X1) = ESC\$+"*m"+STR\$(X1)+"N"
Type:	Graphics, Text
Purpose:	To set the orientation of graphics text.
Remarks:	This function selects the graphics text orientation. This also changes the direction of line feed, carriage return, and back space. The desired orientation is specified by a number defined as: <ul style="list-style-type: none">1 normal text orientation2 rotate 90 degrees counter clockwise3 rotate 180 degrees counter clockwise4 rotate 270 degrees counter clockwise
Example:	SAMPLE6.BAS (on BPL disc)

```
5000 PRINT GSETUP$           'Prepare to use graphics
5010 PRINT FNGTSIZE$(2)      'Set graphics text size
5020 PRINT FNMOVE$(255,195)  'Move pen to screen center
5030 FOR I = 1 TO 4
5040 PRINT FNGTROT$(I):      'Set graphics text orientation
5050 PRINT FNGTLABEL$ ("Graphics Text") 'Print graphics text
5060 NEXT I
5070 PRINT GPDFT$:FNMOVE$(50,50); 'Clean exit from sample
5080 PRINT FNGTLABEL$ ("Press Return to exit.")
5090 INPUT A$
5100 PRINT GCLSS$:ADONS$
```


FNGTSIZE\$(X1)

Statement: DEF FNGTSIZE(X1) = ESC\$ + "*"m" + STR\$(X1) + "M"

Type: Graphics, Text

Purpose: To set the size of graphics text.

Remarks: This function sets the size of graphics text. The vector lists that define the current character set are scaled with this scale factor.

Example: 5000 PRINT FNGTSIZE\$(2);
(See FNGTLABEL\$ for a further example.)

FNLINPAT\$(X1)

Statement: DEF FNLINPAT\$(X1) = ESC\$ + "*"m" + STR\$(X1) + "B"

Type: Graphics, Vector

Purpose: To select the type of line to be used for vectors.

Remarks: The possible line types available are:

- 1 Solid line
- 2 User-defined line (see FNDEFLINES\$)
- 3 Current area fill pattern (see FNFILLPAT\$)
- 4-10 Various combinations of dashes & points
- 11 Point plot

Example: SAMPLE9.BAS (on BPL disc)

```
5000 PRINT GSETUP$           'Prepare to use Graphics
5010 FOR I = 4 TO 11
5020 PRINT FNLINPAT$(I)     'Set line pattern to use
5030 R = I * 20
5040 PRINT FNLYNE$(50,R,500,R) 'Draw line
5050 NEXT I
5060 PRINT GPDFT$;FNMOVES$(50,50) 'clean exit from sample
5070 PRINT FNGTLABEL$ ('Press Return to exit.')
5080 INPUT A$
5090 PRINT GCLSS$,ADONS
```

FNLOCATE\$(R1,C1)

Statement:

```
DEF FNLOCATE$(R1,C1) =  
ESC$+"&a+STR$(R1)+"y"+STR$(C1)+"C"
```

Type:

Alpha, Display

Purpose:

To locate the cursor at a specified row and column on the screen.

Remarks:

This function works the same as the command "Locate" in many other BASICs. The upper left corner of the alpha screen is (0,0) the lower right is (23,79).

It is necessary to follow the command with a semicolon ; to prevent the insertion of a carriage return by BASIC.

Example:

```
5000 PRINT FNLOCATE$(10,22);
```

FNLYNE\$(X1,Y1,X2,Y2)

Statement:

```
DEF FNLYNE$(X1,Y1,X2,Y2) = FNMOVE$(X1,Y1)+  
FNDRAW$(X2,Y2)
```

Type:

Graphics, Vector

Purpose:

To draw a line between any two points.

Remarks:

This function combines two other functions: FNMOVE\$ & FNDRAW\$. A line is drawn between the points (X1,Y1) and (X2,Y2). The line style is that specified by the function FNLINPAT\$.

Example:

SAMPLE10.BAS (on BPL disc)

```
5000 PRINT GSETUP$           'Prepare to use graphics  
5010 PRINT FNLYNE$(0,0,511,389) 'Draw a diagonal line  
5020 PRINT FNLYNE$(0,389,511,0) 'Draw another diagonal line  
5030 PRINT GPDFT$;FNMOVE$(80,50); 'Clean exit from sample  
5040 PRINT FNGTLABEL$ ("Press Return to exit.")  
5050 INPUT A$  
5060 GCLS$;ADON$
```

FNMOVE\$(X1,Y1)

Statement:

```
DEF FNMOVE$(X1,Y1) = ESC$ + "*"pa" + STR$(X1) + "," +  
STR$(Y1) + "Z"
```

Type:

Graphics, Vector

Purpose:

To move the graphics pen to a new position.

Remarks:

This function lifts the graphics pen and moves the pen to the new coordinate position. The pen is lowered at the end of the operation. This function is used to move the pen to a new position in preparation for writing graphics text and drawing a point or line.

Example:

```
5000 PRINT FNMOVE$ (250,250);
```

FNPK16\$(MSB,LSB)

Statement:

```
DEF FNPK16$(MSB,LSB)=CHR$(LSB)+CHR$(MSB)
```

Type:

AGIOS

Purpose:

To create a two-byte string buffer with a bit pattern equivalent to a sixteen bit integer.

Remarks:

The string is created from two eight bit numeric values which make up the integer value, and is stored in BASIC 'MKI\$' format.

This function is used primarily in conjunction with the AGIOS functions, where system calls are specified as eight bit functions followed by eight bit sub-functions.

The AGIOS function in order to generate a two-character escape sequence, requires a sixteen bit word with a zero as the most significant byte, and a 16 in the least significant byte. This word is followed by a one byte ASCII code which determines the function to be performed. To create a buffer to 'home' the cursor, the buffer passed to AGIOS should contain:

```
CHR$(16)+CHR$(0)+"H"
```

The FNPK16\$ function permits you to create this buffer:

```
BUF$=FNPK16$(0,16)+"H"
```

Either form is acceptable: the latter is more clearly consistent with the HP AGIOS documentation.

Example:

```
5000 BUF$=FNPK16$ (0,16) + "H"  
5010 CALL AGIOS (BUF$)
```

FNPSET\$(X1,Y1)

Statement:	DEF FNPSET\$(X1,Y1) = FNMOVE\$(X1,Y1)+POINTPLOT\$
Type:	Graphics, Vector
Purpose:	To draw a point at the specified coordinates.
Remarks:	This function combines the function FNMOVE\$ with the variable POINTPLOT\$ to draw a point at the specified coordinates.
Example:	SAMPLE7.BAS (on BPL disc)

```

5000 PRINT GSETUP$           'Prepare to use graphics
5010 FOR I = 1 TO 500 STEP 5
5020 PRINT FNPSET$(I,200)    'Draw a point every 5 pixels
5030 NEXT I
5040 PRINT GPDFT$;FNMOVE$(50,50); 'clean exit from sample
5050 PRINT FNGTLABEL$ ("Press Return to exit.")
5060 INPUT A$
5070 PRINT GCLSS$;ADON$

```

FNSKD\$(K,L\$,RET\$)

Statement:	DEF FNSKD\$(K,L\$,RET\$) = ESC\$+"&f"+STR\$(K)+"k2a" +STR\$(LEN(L\$))+"d"+STR\$(LEN(RET\$))+"L"+L\$+RET\$
Type:	Alpha, Softkey
Purpose:	To define a user softkey.
Remarks:	This function can be used to define user softkeys. The following conventions need to be followed:

The key number K must be between 1 and 8.

The label L\$ is 0 to 16 characters in length. A length of zero will leave the label unchanged. The first 8 characters will appear on the first line, the next 8 on the second line.

The return string RET\$ is -1 to 80 characters in length. A value of -1 indicates the existing return buffer should be set to a null string. Any other value indicates the number of characters which should be returned to the application when the softkey is pressed.

For further information on programming the softkeys and how parameters of this function can be changed to accomplish different purposes see the softkey section of **Practical Programming Techniques** in this manual.



The "G" and "H" straps in the terminal configuration must be set to YES for softkeys to be used correctly.

Example:

See SAMPLE19.BAS (on BPL disc)

```
5000 PRINT SKOFF$           'Turn off softkeys
5010 PRINT ESC$ + "$s1g1H" 'Set G & H straps
5020 PRINT SKUL$           'Unlock softkey labels
5030 PRINT FNSKD$ (1,"CONTINUE","01") 'Define softkey f1
5040 PRINT FNSKD$ (8," END","08") 'Define softkey f2
5050 PRINT SKON$           'Turn on softkeys
5060 PRINT SKL$           'Lock softkeys on screen
5070 INPUT "Touch a softkey.",A$ 'Wait for softkey input
5080 IF A$="01" THEN 5100
5090 IF A$="08" THEN 5100
5100 PRINT SKUL$           'Unlock softkey labels
5110 PRINT SKM$           'Switch to modes softkeys
5120 PRINT ESC$ + "$s0g0H" 'Set G & H to system defaults
```

FNSOFF\$(X1)

Statement:

```
DEF FNSOFF$(X1)=CHR$(PEEK(X1+1))  
+CHR$(PEEK(X1+2)) interpreted BASIC only  
DEF FNSOFF$(X1)=CHR$(PEEK(X1+2))  
+CHR$(PEEK(X1+3)) 'compiled BASIC only
```

Type:

AGIOS

Purpose:

To generate the address of a character string in memory from the string descriptor block used by BASIC.

Remarks:

The VARPTR function in BASIC can be used to return the address of a string descriptor block for a particular string (CHARBUF\$ in the example below). Function FNSOFF\$ returns a two-character packed buffer containing the actual address of the character string. Normally the string buffer contains information required by an AGIOS.

Example:

```
5000 BUF$="CHARBUF"  
5010 X1=VARPTR (BUF$)  
5020 B$=FNSOFF$ (X1)
```

FNSOFFL\$(X1)

Statement:

```
DEF FNSOFFL$(X1)=FNSOFF$(X1)+DS$
```

Type:

AGIOS

Purpose:

To generate a fully qualified offset and data segment address for use in an AGIOS call.

Remarks:

This function generates a buffer containing a fully qualified address consisting of a sixteen bit offset (using FNSOFF\$) and a sixteen bit segment register value stored in DS\$ during start-up processing by BPL. For several AGIOS functions, the buffer address must be fully qualified. This function can be used for those calls.

Example:

```
5000 BUF$="CHARBUF"  
5010 X1=VARPTR (BUF$)  
5020 BX$=FNSOFFL$ (X1)
```

FNTF\$(R1,R2,C1,C2,RET\$)

Statement: `DEF FNTF$(R1,R2,C1,C2,RET$) = ESC$+“-zg”+STR$(R1)
+“,”+ STR$(R2)+“r”+STR$(C1)+“,”+
STR$(C2)+“c0p1b10e2f2m1a”+STR$(LEN(RET$))
+“L”+RET$`

Type: Touch, Display

Purpose: To define touch fields of various sizes.

Remarks: This function defines a touch field between the rows R1 and R2 and the columns C1 and C2. RET\$ contains the string to be returned when the field is touched. A further explanation of each possible parameter can be found under touch screen information in the **Practical Programming** (Chapter 3) of this manual.

WARNING: R1 must be less than R2
C1 must be less than C2

Example: `5000 Print FNTF$ (2,6,2,10,“GOOD TOUCH MY FRIEND”);`
(See FNTOUCH\$ for a further example)

FNTFDEL\$(R1,C1)

Statement: `DEF FNTFDEL$(R1,C1) = ESC$+“-zd”
+STR$(R1)+“r”+STR$(C1)+“C”`

Type: Touch, Display

Purpose: To delete a specific touch field.

Remarks: This function deletes the touch field that has as its upper left coordinate (R1,C1). It is good programming practice to delete touch fields when you are finished with them. This can be accomplished by using this function or TSDEL\$.

Example: `5000 PRINT FNTFDEL$ (2,6);`

FNTOUCH\$(R1,C1,RET\$)

Statement:

```
DEF FNTOUCH$(R1,C1,RET$) = ESC$+"-zg"+STR$(R1)
+" "+STR$(R1+1)+"r"+STR$(C1)+" "+STR$(C1+7)
+"c0p1b10e2f2m1a"+STR$(LEN(RET$))+ "L"+RET$
```

Type:

Touch, Display

Purpose:

To define a 2 row by 8 column touch field.

Remarks:

See Chapter 3 'Practical Programming'.

Example:

SAMPLE13.BAS (on BPL disc)

```
5000 'Define a 4 X 4 grid of touch fields
5010 'with return values of 1 to 16.
5020 PRINT CLS$;
5030 K=0 'Setup touch field parameters
5040 FOR R1 = 3 to 15 STEP 4
5050 FOR C1 = 21 TO 51 STEP 10
5060 K = K + 1
5070 PRINT
FNTF$(R1,R1+2,C1,C1+7,STR$(K)) 'Define touch field
5080 NEXT C1
5090 NEXT R1
5100 PRINT TSON$ 'Turn on touch screen
5110 PRINT FNLOCATE$(20,0);
5120 PRINT "Touch a square to see a return value, report is on release."
5130 PRINT "Press Return to exit."
5140 A$=INPUT$(1)
5150 IF A$=CR$ THEN 5170 ELSE 5160
5160 PRINT A$+" ":GOTO 5140
5170 PRINT TSDDEL$ 'Delete all touch fields
```

FNUP\$(BUF\$)

Statement:

```
DEF FNUP$(BUF$) = CHR$(ASC(BUF$)+(32*((ASC(BUF$)>96  
AND ASC(BUF$)<123))))
```

Type:

Alpha, Edit

Purpose:

To ensure that a character is uppercase.

Remarks:


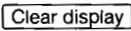
It is often desirable to permit the user to enter a response in either upper or lower case. By using this function, you can 'upper case' whatever is entered, making the test for a match easier.

Example:

```
5000 A$=FNUP$(INPUT$(1))
```



FNXBAT\$(A\$,X1)

Statement:	DEF FNXBAT\$(A\$,X1) = FNPk16\$(0,0) + MKI\$(LEN(A\$)) + FNSOFFL\$(X1)								
Type:	General, AGIOS								
Purpose:	To perform 'batch' AGIOS commands.								
Remarks:	<p>FNXBAT\$ can be used to create a buffer of one or more AGIOS commands which are to be executed at the same time.</p> <p>You must pass FNXBAT\$ the command string as well as the address of the string (which cannot be generated within a function).</p> <p>For example, you will often want to perform a  followed by  to erase the entire screen. To do this, you will need to set up a batch or 'macro' AGIOS command string.</p> <p>Once you have created this string (XCLS\$) you will only need to store it and call it when needed. If one or more of your batched commands require variable parameters (such as FNXMovE\$) you will need to repeat these steps prior to each call.</p>								
Example:	<table><tr><td>5000 BUF\$ = XHOME\$ + XCLR\$</td><td>'Build buffer of commands</td></tr><tr><td>5010 B = VARPTR (BUF\$)</td><td>'Find its address</td></tr><tr><td>5020 XCLS\$ = FNXBAT\$ (BUF\$,B)</td><td>'Build Batch Buffer</td></tr><tr><td>5030 CALL AGIOS (XCLS\$)</td><td>'Call Batch Function</td></tr></table>	5000 BUF\$ = XHOME\$ + XCLR\$	'Build buffer of commands	5010 B = VARPTR (BUF\$)	'Find its address	5020 XCLS\$ = FNXBAT\$ (BUF\$,B)	'Build Batch Buffer	5030 CALL AGIOS (XCLS\$)	'Call Batch Function
5000 BUF\$ = XHOME\$ + XCLR\$	'Build buffer of commands								
5010 B = VARPTR (BUF\$)	'Find its address								
5020 XCLS\$ = FNXBAT\$ (BUF\$,B)	'Build Batch Buffer								
5030 CALL AGIOS (XCLS\$)	'Call Batch Function								

FNXDRAW\$(X1,Y1)

Statement: $\text{DEF FNxDRAW}\$(X1,Y1) = \text{FNPK16}\$(4,44) + \text{MKI}\$(X1) + \text{MKI}\$(Y1)$

Type: Graphics, Vector, AGIOS

Purpose: To draw a line from the current pen position.

Remarks: FNxDRAW\$ works just as FNDRAW\$ to draw a line from the current graphics pen position to the specified coordinate position.

Example: $\text{BUF}\$ = \text{FNxDRAW}\$(100,100); \text{CALL AGIOS}(\text{BUF}\$)$

FNXDRAWMODE\$(X1)

Statement: $\text{DEF FNxDRAWMODE}\$(X1) = \text{FNPK16}\$(4,17) + \text{MKI}\$(X1)$

Type: Graphics, Display, AGIOS

Purpose: To select the vector drawing mode.

Remarks: FNxDRAWMODE\$ works just as FNDRAWMODE\$ to select the vector drawing mode. Refer to the description under FNDRAWMODE\$ for additional information.

Example: $\text{BUF}\$ = \text{FNxDRAWMODE}\$(2) : \text{CALL AGIOS}(\text{BUF}\$)$

FNXENHON\$

Statement:
$$\text{DEF FNXENHON}(\text{BUF}\$) = \text{FNPk16}\$(0,18) + \text{CHR}\$(0) + \text{MID}\$(\text{BUF}\$,1,1)$$

Type: Alpha, Display, AGIOS

Purpose: To define the type of enhancement for characters which follow on the current line.

Remarks: FNXENHON\$ works just as FNENHON\$ to define the alpha display enhancement for characters printed to the right of the cursor when this call is made. The enhancement remains enabled until it is turned off (see XENHOFF\$ and ENHOFF\$) or until the cursor is moved to a new line.

Example:
$$\text{BUF}\$ = \text{FNXENHON}\$ ("B") : \text{CALL AGION}(\text{BUF}\$)$$

FNXFILLPAT\$(X1)

Statement:
$$\text{DEF FNXFILLPAT}\$(\text{X1}) = \text{FNPk16}\$(4,23) + \text{MKI}\$(\text{X1})$$

Type: Graphics, Polygon, AGIOS

Purpose: To select the desired fill pattern for polygon fill.

Remarks: FNXFILLPAT\$ works just as FNFILLPAT\$ to select the pattern which is active for polygon fill operations.

Example:
$$\text{BUF}\$ = \text{FNXFILLPAT}\$(1) : \text{CALL AGIOS}(\text{BUF}\$)$$

FNXGCMA\$(X1,Y1)

Statement:
$$\text{DEF FNXGCMA}\$(\text{X1},\text{Y1}) = \text{FNPk16}\$(4,11) + \text{MKI}\$(\text{X1}) + \text{MKI}\$(\text{Y1})$$

Type: Graphics, Display, AGIOS

Purpose: To move the graphics cursor to the specified position.

Remarks: FNXGCMA\$ works just as FNGCMA\$ to move the absolute graphics screen coordinate specified and occurs whether the cursor is actually displayed or not.

Example:
$$\text{BUF}\$ = \text{FNXGCMA}\$(0,0) : \text{CALL AGIOS}(\text{BUF}\$)$$

FNXGCMR\$(X1,Y1)

Statement: DEF FNXGCMR\$(X1,Y1) = FNPk16\$(4,12) + MKI\$(X1) +
MKI\$(Y1)

Type: Graphics, Display, AGIOS

Purpose: To move the graphics cursor relative to the current graphics cursor position.

Remarks: FNXGCMR\$ works just as FNGMCR\$ to move the graphics cursor relative to the current position

Example: BUF\$ = FNXGCMR\$ (25,-25) : CALL AGIOS (BUF\$)

FNXGTORGN\$(X1)

Statement: DEF FNXGTORGN\$(X1) = FNPk16\$(4,33) + MKI\$(X1)

Type: Graphics, Text, AGIOS

Purpose: To set the origin of graphics text characters.

Remarks: FNXGTORGN\$ works just as FNGTORGN\$ to set the justification for graphics text within each character cell.

Example: BUF\$ = FNXGTORGN\$ (0) : CALL AGIOS (BUF\$)

FNXGTROT\$(X1)

Statement: DEF FNXGTROT\$(X1) = FNPk16\$(4,30) + MKI\$(X1)

Type: Graphics, Text, AGIOS

Purpose: To set the orientation of graphics text.

Remarks: FNXGTROT\$ works just as FNGTROT\$ to set the orientation of graphics text.

Example: BUF\$ = FNXGTROT\$ (3) : CALL AGIOS (BUF\$)

FNXGTSIZE\$(X1,X2,Y1,Y2)

Statement: DEF FNXGTSIZE\$(X0,X1,Y0,Y1) = FNPk16\$(4,29) +
FNPk16\$(X2,X1) +
FNPk16\$(Y2,Y1)

Type: Graphics, Text, AGIOS

Purpose: To set the size of graphics text.

Remarks: FNXGTSIZE\$ is very different from FNGTSIZE\$ in that non-integral text sizes may be specified. The format of each parameter is:

Example: BUF\$ = FNXGTSIZE\$ (1,5,1,0) : CALL AGIOS (BUF\$)

FNXLINPAT\$(X1)

Statement: FNXLINPAT\$(X1) = FNPk16\$(4,18) + CHR\$(X1)

Type: Graphics, Vector, AGIOS

Purpose: To select the line pattern to be used in vector draw.

Remarks: FNXLINPAT\$ works just as FNLINPAT\$ to select any of 11 possible line patterns to be used in vector drawing operations. Refer to FNLINPAT\$ for more information on the possible line types.

Example: XBUF\$ = FNXLINPAT\$ (4):CALL AGIOS (XBUF\$)

FNXLOCATE\$(R1,C1)

Statement:

```
DEF FNXLOCATE$(R1,C1) = FNPk16$(0,17) + CHR$(153) +  
                        MKI$(C1) + MKI$(R1)
```

Type:

Alpha, Display, AGIOS

Purpose:

To position the alpha cursor.

Remarks:

FNXLOCATE\$ works just as FNLOCATE\$ to position the alpha cursor at a particular row and column location on the screen. This function positions the cursor to a fixed location on the display screen.

Example:

```
BUF$ = FNXLOCATE$ (10,10) : CALL AGIOS (BUF$)
```

FNXMOVE\$(X1,Y1)

Statement:

```
DEF FNXMOVE$(X1,Y1) = FNPk16$(4,40) + MKI$(X1) +  
                        MKI$(Y1)
```

Type:

Graphics, Vector, AGIOS

Purpose:

To move the graphics cursor to a new position.

Remarks:

FNXMOVE\$ works just as FNMOVE\$ to position the graphics cursor to an absolute position on the display without drawing the line, and puts the pen down at the end of the move.

Example:

```
BUF$ = FNXMOVE$ (511,389) : CALL AGIOS (BUF$)
```


FNXTCESC\$(A\$)

Statement: FNXTCESC\$(A\$) = FNPk16\$(0,16) + MID\$(A\$,1,1)

Type: General, AGIOS

Purpose: To execute a two-character escape sequence.

Remarks: FNXTCESC\$ allows you to execute any two character escape sequence supported by the HP Touchscreen PC. Refer to the *HP Touchscreen PC Terminal User's Guide* for information on the actual escape sequences.

Example: 5000 BUF\$ = FNXTCESC\$ ("A") : CALL AGIOS (BUF\$)

GCL\$

Statement: ESC\$+""dA"

Type: Graphics, Display

Purpose: To clear the graphics display.

Remarks: This function clears the entire 390 X 512 graphics display setting all pixels to 0 or off.

Example: 5000 PRINT GCL\$;

GCOFF\$

Statement: ESC\$+""dL"

Type: Graphics, Display

Purpose: To turn off the graphics cursor.

Remarks: Turning off the graphics cursor does not affect graphics memory.

Example: 5000 PRINT GCOFF\$;

GCONS

Statement: ESC\$+""*dK"
Type: Graphics, Display
Purpose: To turn on the graphics cursor.
Remarks: Turning on the graphics cursor does not affect the data in graphics memory.
Example: 5000 PRINT GCONS;

**GDFT\$**

Statement: ESC\$+""*mR"
Type: Graphics, Display
Purpose: To set graphics parameters to their default values.
Remarks: The default values for the graphics parameters are:

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

Example: 5000 PRINT GDFT\$;

GDOFF\$

Statement: `GDOFF$ = ESC$+""*dD"`

Type: Graphics, Display

Purpose: To turn off graphics display.

Remarks: Turning off the graphics display does not affect graphics display memory. This means that if graphics display is turned off and then back on it will be the same.

Example: `5000 PRINT GDOFF$;`

GDON\$

Statement: `GDON$ = ESC$+""*dC"`

Type: Graphics, Display

Purpose: To turn on the graphics display.

Remarks: Turning on the graphics display does not affect the graphics memory.

Example: `5000 PRINT GDON$;`

GPDFT\$

Statement: `GPDFT$ = ESC$+""*m1R"`

Type: Graphics, Display

Purpose: To set the graphics picture defaults.

Remarks: The default values of the graphics picture definition parameters are:

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

Example: `5000 PRINT GPDFT$;`

GRESET\$

Statement: `GRESET$ = ESC$+""*wR"`

Type: Graphics, Display

Purpose: To reset all graphics parameters to the power on state.

Example: `5000 PRINT GRESET$;`

GSET\$

Statement:	GSET\$ = ESC\$+""*dB"
Type:	Graphics, Display
Purpose:	To set graphics memory to inverse video.
Remarks:	The entire 512 x 390 graphics display is set to 1 or on.
Example:	5000 PRINT GSET\$; (See POLYGON\$ for a further example.)

GSETUP\$

Statement:	GSETUP\$ = GCLS\$+GDON\$+ADOFF\$
Type:	Graphics, Display
Purpose:	To prepare the screen for using graphics.
Remarks:	<p>This is an example of how several library variables can be combined to accomplish multiple tasks with a single statement. The graphics screen is cleared (GCLS\$), the graphics display is turned on (GDON\$), and the alpha display is turned off (ADOFF\$). The contents of alpha display memory is not affected.</p> <p>This variable is often printed in a program just prior to the first use of graphics.</p>
Example:	5000 PRINT GSETUP\$;

GTOFF\$

Statement:

GTOFF\$ = ESC\$+""*dT"

Type:

Graphics, Text

Purpose:

To turn off graphics text.

Remarks:

By turning off the graphics, text mode characters will go to the alpha display.

Example:

5000 PRINT GTOFF\$;

GTON\$

Statement:

GTON\$ = ESC\$+""*dS"

Type:

Graphics, Text

Purpose:

To turn on graphics text mode.

Remarks:

Characters that normally go to the alphanumeric display will go to the graphics display when graphics text mode is on.



Depending on graphics text origin the text may not appear until after **Return** . See FNGTORGN\$.

Example:

5000 PRINT GTON\$;

GTSOFF\$

Statement: `GTSOFF$ = ESC$+""*mP"`

Type: Graphics, Text

Purpose: To turn off slanted graphics text.

Remarks: Once graphics text slant is turned off graphics text characters will be output normally.

Example: `5000 PRINT GTSOFF$;`

GTSON\$

Statement: `GTSON$ = ESC$+""*mO"`

Type: Graphics, Text

Purpose: To turn on slanted graphics text.

Remarks: The slant that is used for graphics text characters is 26.57 degrees.

Example: `5000 PRINT GTSON$;`

HOMES\$

Statement: `HOMES$ = ESC$+""H"`

Type: Alpha, Display

Purpose: To move the alpha cursor to the home position.

Remarks: The home position is row 1 column 1 or in the function `FNLOCATE$(0,0)`. Be sure to follow `HOMES$` with a semicolon to prevent BASIC from issuing the carriage return.

Example: `5000 PRINT HOMES$;`

HOMEDWN\$

Statement: HOMEDWN\$ = ESC\$+"F"

Type: Alpha, Display

Purpose: To move the alpha cursor to the bottom of screen memory contents.

Remarks: This function will move the cursor to the end of the contents of screen memory, not the bottom or 48th row of screen memory. HOMEDWN\$ will only move the cursor the 48th row if the entire alpha memory is full.

Example: 5000 PRINT HOMEDWN\$;

IL\$

Statement: IL\$ = ESC\$+"L"

Type: Alpha, Edit

Purpose: To programmatically insert a line in alpha memory.

Remarks: This function works the same as pressing on the keyboard.

Example: 5000 PRINT IL\$;

KBOFF\$

Statement: KBOFF\$ = ESC\$+"c"

Type: General

Purpose: To turn off the keyboard.

Remarks: This function will lock the keyboard, preventing the user from inputting information from the keyboard. When using this function be sure to turn the keyboard back on programmatically or the user will have to perform a soft reset to get going again.

Example: 5000 PRINT KBOFF\$;

KBON\$

Statement: KBON\$ = ESC\$+"b"
Type: General
Purpose: To turn on the keyboard.
Remarks: This function turns on the keyboard to allow user input.
Example: 5000 PRINT KBON\$;

MEML\$

Statement: MEML\$ = ESC\$+"I"
Type: Alpha, Display
Purpose: To lock a portion of screen memory.
Remarks: This function will lock the alpha screen memory from the cursor position to the top of the displayed screen so that this portion of memory will not be scrolled off the top of the screen. To use it, move the cursor to the desired row and then print MEML\$. This has the same effect as the memory lock system softkey. It is very useful for locking instructions or touch fields at the top of the screen.
Example: SAMPLE11.BAS (on **BPL** disc)

```
5000 PRINT CLS$           'Clear Screen
5010 PRINT FNLOCATE$ (0,0); 'Move to first roll
5020 PRINT FNENHON$ ("J"); 'Use half bright inverse video
5030 PRINT STRING$ (80," ") 'Show enhancement for entire line
5040 PRINT FNLOCATE$ (0,0);
5050 PRINT FNCENTER$ ("BANNER LINE",80) 'Center the title
5060 PRINT MEML$         'Set Memory Lock
5070 PRINT "Please try to scroll the banner line off the screen."
5080 PRINT "Press Shift and up arrow or down arrow to scroll."
5090 PRINT "Next press softkey f6 or type 'PRINT MEMUL$' to unlock memory"
5100 PRINT "and allow the banner line to scroll."
5110 PRINT:PRINT:PRINT
```

MEMUL\$

Statement: MEMUL\$ = ESC\$+"m"
Type: Alpha, Display
Purpose: To unlock screen memory.
Remarks: Place the cursor on the row where memory lock (MEML\$) was previously used and then print this function.
Example: 5000 PRINT MEMUL\$;

ORGNSET\$

Statement: ORGNSET\$ = ESC\$+"*pE"
Type: Graphics, Origin
Purpose: To set the relocatable origin to the current pen position.
Example: 5000 PRINT ORGNSET\$;
(See FNBOXFR\$ for a further example)

ORGNSETCP\$

Statement: ORGNSETCP\$ = ESC\$+"*mL"
Type: Graphics, Origin
Purpose: To set the relocatable origin to the current graphics cursor position.
Example: 5000 PRINT ORGNSETCP\$;

ORGNSETPP\$

Statement: ORGNSETPP\$ = ESC\$+""*mK"

Type: Graphics, Origin

Purpose: To set the relocatable origin to the current pen position.

Remarks: Several graphics commands can be issued relative to the relocatable origin rather than the absolute origin of (0,0) in the lower left corner. Setting the relocatable origin may provide advantages to the application program.

Example: 5000 PRINT ORGNSETPP\$;

PCMOVE\$

Statement: PCMOVE\$ = PENUP\$+ESC\$+""*pC"

Type: Graphics, Vector

Purpose: To move the pen to the current graphics cursor position.

Example: 5000 PRINT PCMOVE\$;

PCPLOT\$

Statement: PCPLOT\$ = PENDWN\$+ESC\$+""*pC"

Type: Graphics, Vector

Purpose: To draw a line to the current graphics cursor position.

Example: 5000 PRINT PCPLOT\$;

PENDWN\$

Statement: `PENDWN$ = ESC$+“*pB”`

Type: Graphics, Vector

Purpose: To lower the graphics pen.

Example: `5000 PRINT PENDWN$;`

PENUP\$

Statement: `PENUP$ = ESC$+“*pA”`

Type: Graphics, Vector

Purpose: To raise the graphics pen.

Example: `5000 PRINT PENUP$;`

POINTPLOT\$

Statement: `POINTPLOT$ = ESC$+“*pD”`

Type: Graphics, Vector

Purpose: To draw one point.

Remarks: A point will be drawn at the current pen position and then the pen is raised.

Example: `5000 PRINT POINTPLOT$`
(See FNPSET\$ for a further example)

POLYGON\$

Statement:	POLYGON\$ = ESC\$+"*ps"
Type:	Graphics, Polygon
Purpose:	To start polygon area fill definition.
Remarks:	<p>Starts the definition of a polygon for area fill. The boundary pen is lowered with the use of this function.</p> <p>To use this function print the variable POLYGON\$ then a + and a " (quote mark). Next, specify all of the (X,Y) coordinates and that define the polygon. The sequence must end with an upper case 'T'. Be sure the first and last coordinates are the same so the polygon is closed. See line 5120 for an example of this function.</p>

Example:

SAMPLE3.BAS (on BPL disc)

```
5000 PRINT GSETUP$           'Prepare to use graphics
5010 PRINT FNDRAWMODE$(3)    'Select complement mode
5020 PRINT FNMOVE$(0,0)      'Locate graphics pen
5030 PRINT GSET$            'Graphics to inverse video
5040 PRINT FNMOVE$(10,130)
5050 '                        Polygon area to open window
5060 PRINT POLYGON$+"10,130,500,130,500,260,10,260,10,130T"
5070 PRINT FNMOVE$(14,134)
5080 '                        Frame in window
5090 PRINT FNBOX$(14,134,496,256)
5100 PRINT FNMOVE$(28,250)
5110 '                        Polygon area for T
5120 PRINT POLYGON$+"28,250,178,250,178,230,118,230,118,140,88,140,88,230,28,230,28,250T"
5130 PRINT FNMOVE$(190,250)
5140 '                        Polygon area for L
5150 PRINT POLYGON$+"190,250,220,250,220,160,310,160,310,140,190,140,190,250T"
5160 PRINT FNMOVE$(390,250)
5170 '                        Polygon areas for A
5180 PRINT POLYGON$+"390,250,420,250,480,140,450,140,427,185,383,185,360,140,330,140,390,250T"
5190 PRINT FNMOVE$(405,230)
5200 PRINT POLYGON$+"405,230,418,205,392,205,405,230T"
5210 PRINT FNMOVE$(425,240)
5220 PRINT FNGTLABEL$("TM")
5230 PRINT GPDLTFT$
5240 PRINT FNMOVE$(10,10);FNGTLABEL$("Press Return to exit.")
5250 INPUT:A$                'Wait for a key stroke to continue
5260 PRINT GCLS$,ADONS       'Clear graphics & return to alpha screen
```

RBLOFF\$

Statement: RBLOFF\$ = ESC\$+""dN"
Type: Graphics, Vector
Purpose: To turn off the rubber band line.
Example: 5000 PRINT RBLOFF\$;

RBLON\$

Statement: RBLON\$ = ESC\$+""dM"
Type: Graphics, Vector
Purpose: To turn on the rubber band line.
Remarks: Also turns on the graphics cursor.
Example: SAMPLE12.BAS (on **BPL** disc)

```
5000 PRINT GSETUP$           'Prepare to use Graphics
5010 PRINT FNGCMA$ (10,385)  'Move graphics cursor
5020 PRINT FNGTSIZE$ (1)    'Set graphics text size
5030 PRINT GTON$            'Turn on graphics text mode
5040 PRINT "Please use graphics cursor keys to move the graphics cursor."
5050 PRINT "This will show how the rubber band line works."
5060 PRINT "You may need to press 'CTRL' and '-' on the graphics pad"
5070 PRINT "to activate the graphics cursor keys."
5080 PRINT "Press Return to exit."
5090 PRINT GTOFF$           'Turn off graphics text mode
5100 PRINT FNMOVE$ (256,195) 'Move graphics pen
5110 PRINT RBLON$          'Turn on rubber band line
5120 PRINT FNGCMA$ (200,100) 'Move graphics cursor
5130 PRINT GPDFT$          'Clean exit from sample
5140 INPUT A$
5150 PRINT RBLOFF$;GCOFF$;GCLS$;ADON$
```

SKL\$

Statement:	SKL\$ = ESC\$ + "&jS"
Type:	Alpha, Softkey
Purpose:	To lock the softkey labels.
Remarks:	Locks the softkey labels to their current state. If the user labels are defined and on the screen this function will keep the user from changing the labels. It will also keep the labels locked off the screen.
Example:	5000 PRINT SKL\$; (See FNSKD\$ for a further example.)

SKM\$

Statement:	SKM\$ = Esc\$ + "&jA"
Type:	Alpha, Softkey
Purpose:	To switch to the modes softkey set.
Remarks:	This set of keys includes: Line Modify Modify All Block Mode Remote Mode Smooth Scroll Memory Lock Display Functions Auto LF
Example:	5000 PRINT SKOFF\$; (See FNSKD\$ for a further example.)



SKOFF\$

Statement: SKOFF\$ = ESC\$ + "&j@"

Type: Alpha, Softkey

Purpose: To turn off the softkey labels.

Remarks: Also turns off the status line and cursor position indicator.

Example: 5000 PRINT SKOFF\$;
(See FNSKD\$ for a further example.)

SKON\$

Statement: SKON\$ = ESC\$ + "&jB"

Type: Alpha, Softkey

Purpose: To turn on the softkey labels.

Remarks: The last set of softkeys displayed or requested will be shown.

Example: 5000 PRINT SKON\$;
(See FNSKD\$ for a further example.)

SKUL\$

Statement: SKUL\$ = ESC\$ + "&jR"

Type: Alpha, Softkey

Purpose: To unlock the softkey labels so they may be changed.

Remarks: This function may be used with the softkey labels either on or off.

Example: 5000 PRINT SKUL\$;
(See FNSKD\$ for a further example.)

TAB\$

Statement: TAB\$ = ESC\$+"I"
Type: Alpha, Edit
Purpose: To move the cursor to the next defined tab stop.
Remarks: Similar to Tab on the keyboard.
Example: 5000 PRINT TAB\$;

TSDEL\$

Statement: TSDEL\$ = ESC\$+"-zD"
Type: Touch, Display
Purpose: To delete all defined touch fields.
Remarks: Touch screen fields remain active until specifically deleted. This variable will delete all currently defined fields. Individual fields may be deleted by using FNTFDEL\$.
Example: 5000 PRINT TSDEL\$;

TSOFF\$

Statement: TSOFF\$ = ESC\$ + "-z0N"
Type: Touch, Display
Purpose: To turn off the touch screen.
Remarks: Touch fields remain defined but can not be activated until the touch screen is turned on.
Example: 5000 PRINT TSOFF\$;

TSO\$

Statement: TSON\$ = ESC\$+“-z2n1A”

Type: Touch, Display

Purpose: To turn on the Touch Screen.

Remarks: Specifies ASCII reporting of touch fields only.

Example: 5000 PRINT TSON\$;

TSRESET\$

Statement: TSRESET\$ = ESC\$+“-zJ”

Type: Touch, Display

Purpose: To reset the touch screen to the default parameters.

Example: 5000 PRINT TSRESET\$;

XADOFF\$

Statement: XADOFF\$ = FNPk16\$(4,6)

Type: Alpha, Display, AGIOS

Purpose: To turn off the alpha display.

Remarks: XADOFF\$ works just as ADOFF\$ to turn off the alpha display without affecting the contents of screen memory.

Example: 5000 CALL AGIOS (XADOFF\$)

XADON\$

Statement: XADON\$ = FNPk16\$(4,5)

Type: Alpha, Display, AGIOS

Purpose: To turn on the alpha display.

Remarks: XADON\$ works just as ADON\$ to turn on the alpha display without affecting the contents of screen memory.

Example: 5000 CALL AGIOS (XADON\$)

XBDYPENOFF\$

Statement: XBDYPENOFF\$ = FNPk16\$(4,25)

Type: Graphics, Polygon, AGIOS

Purpose: To turn off the drawing of a boundary pen around a polygon.

Remarks: XBDYPENOFF\$ works just as BDRYPENOFF\$ to turn off automatic boundary pen drawing.

Example: 5000 CALL AGIOS (XBDYPENOFF\$)

XBDYPENON\$

Statement: XBDYPENON\$ = FNPk16\$(4,24) + MKI\$(1)

Type: Graphics, Polygon, AGIOS

Purpose: To select the pen to draw a polygon boundary.

Remarks: XBDYPENON\$ works just as BDRYPENON\$ to draw the boundary of a filled polygon.

Example: 5000 CALL AGIOS (XBDYPENON\$)

XB TAB\$

Statement: $\text{XB TAB\$} = \text{FNP K16\$}(0,16) + \text{"i"}$

Type: Alpha, Edit

Purpose: To move the cursor back to the previous tab stop.

Remarks: XB TAB\$ works just as BT AB\$ to move the alpha cursor back to the previous tab stop or, if none are set, to the left margin.

Example: 5000 CALL AGIOS (XB TAB\$)

XCLL\$

Statement: $\text{XCLL\$} = \text{FNP K16\$}(0,16) + \text{"K"}$

Type: Alpha, Edit

Purpose: To clear the current line in alpha memory.

Remarks: XCLL\$ works just as CLL\$ to clear the alpha display from the current cursor position to the end of the current line.

Example: 5000 CALL AGIOS (XCLL\$)

XCLR\$

Statement: $\text{XCLR\$} = \text{FNP K16\$}(0,16) + \text{"J"}$

Type: Alpha, Edit, AGIOS

Purpose: To clear alpha display to the end of screen.

Remarks: XCLR\$ works just as CLR\$ to clear alpha display memory from the current cursor position to the bottom of display memory.

Example: 5000 CALL AGIOS (XCLR\$)

XCOFF\$

Statement: XCOFF\$ = FNPk16\$(4,14)

Type: Alpha, Display, AGIOS

Purpose: To turn off the alpha cursor.

Remarks: XCOFF\$ works just as COFF\$ to turn off the alpha cursor.

Example: 5000 CALL AGIOS (XCOFF\$)

XCON\$

Statement: XCON\$ = FNPk16\$(4,13)

Type: Alpha, Display, AGIOS

Purpose: To turn on the alpha cursor.

Remarks: XCON\$ works just as CON\$ to turn on the alpha cursor.

Example: 5000 CALL AGIOS (XCON\$)

XDL\$

Statement: XDL\$ = FNPk16\$(0,16) + "M"

Type: Alpha, Edit

Purpose: To delete the line the cursor is on.

Remarks: XDL\$ works just as DL\$ to delete the line in screen memory and scrolls any lines below that line 'up'.

Example: 5000 CALL AGIOS (XDL\$)

XENHOFF\$

Statement: XENHOFF\$ = FNPk16\$(0,18) + CHR\$(0) + "@"

Type: Alpha, Display, AGIOS

Purpose: To turn off display enhancements.

Remarks: XENHOFF\$ works just as ENHOFF\$ to turn off display enhancements on the current line from the current cursor position to the end of line (or to the next location where an enhancement is turned on). Note that the corresponding call to turn on enhancements is a function which must be assigned to a string which is then passed to AGIOS.

XENHOFF\$ is a special case of FNxENHON\$.

Example: 5000 CALL AGIOS (XENHOFF\$)

XGCLS\$

Statement: XGCLS\$ = FNPk16\$(4,1)

Type: Graphics, Display, AGIOS

Purpose: To clear the graphics display.

Remarks: XGCLS\$ works just as GCLS\$ to clear the entire graphics display by setting all pixels to 0 or off.

Example: 5000 CALL AGIOS (XGCLS\$)

XGCOFF\$

Statement: XGCOFF\$ = FNPk16\$(4,8)

Type: Graphics, Display, AGIOS

Purpose: To turn off the graphics cursor.

Remarks: XGCOFF\$ works just as GCOFF\$ to turn off the graphics cursor. This does not affect the contents of graphics memory.

Example: 5000 CALL AGIOS (XGCOFF\$)

XGCON\$

Statement: XGCON\$ = FNPk16\$(4,7)

Type: Graphics, Display, AGIOS

Purpose: To turn on the graphics cursor.

Remarks: XGCON\$ works just as GCON\$ to turn on the graphics cursor. This does not affect the contents of graphics memory nor does it assume the graphics display is on.

Example: 5000 CALL AGIOS (XGCON\$)

XGDFT\$

Statement: XGDFT\$ = FNPk16\$(4,38)

Type: Graphics, Display, AGIOS

Purpose: To set graphics parameters to their default values.

Remarks: XGDFT\$ works just as GDFT\$ to set graphics parameters to their default values. Refer to GDFT\$ for a review of those default states.

Example: 5000 CALL AGIOS (XGDFT\$)

XGDOFF\$

Statement: XGDOFF\$ = FNPk16\$(4,4)

Type: Graphics, Display, AGIOS

Purpose: To turn off graphics display.

Remarks: XGDOFF\$ works just as GDOFF\$ to turn off graphics memory. It doesn't affect the contents of graphics memory, but it does cause the contents of graphics to become 'invisible'. The graphics cursor is also turned off.

Example: 5000 CALL AGIOS (XGDOFF\$)

XGDON\$

Statement: XGDON\$ = FNPk16\$(4,3)

Type: Graphics, Display, AGIOS

Purpose: To turn on graphics display.

Remarks: XGDON\$ works just as GDON\$ to turn on the graphics display. If there is something in graphics memory it will become 'visible'. If the graphics cursor is 'on', it too will become visible.

Example: 5000 CALL AGIOS (XGDON\$)

XGPDFT\$

Statement: XGPDFT\$ = FNPk16\$(4,72)

Type: Graphics, Display, AGIOS

Purpose: To set graphics picture defaults.

Remarks: XGPDFT\$ works just as GPDFT\$ to set the default picture values. Refer to GPDFT\$ for a review of the default values.

Example: 5000 CALL AGIOS (XGPDFT\$)

XGRESET\$

Statement: XGRESET\$ = FNPk16\$(4,73)

Type: Graphics, Display, AGIOS

Purpose: To set graphics parameters to their power-on state.

Remarks: XGRESET\$ works just as GRESET\$ to set all graphics parameters to their default power-on states.

Example: 5000 CALL AGIOS (XGRESET\$)

XGSET\$

Statement: XGSET\$ = FNPk16\$(4,2)

Type: Graphics, Display, AGIOS

Purpose: To set graphics display to inverse video.

Remarks: XGSET\$ works just as GSET\$ to set every pixel in graphics memory to 1 or on.

Example: 5000 CALL AGIOS (XGSET\$)

**XGTOFF\$**

Statement: XGTOFF\$ = FNPk16\$(4,16)

Type: Graphics, Text, AGIOS

Purpose: To turn off graphics text mode.

Remarks: XGTOFF\$ works just as GTOFF\$ to turn off graphics text mode. Standard console output will be directed to alpha memory.

Example: 5000 CALL AGIOS (XGTOFF\$)

XGTON\$

Statement: XGTON\$ = FNPk16\$(4,15)

Type: Graphics, Text, AGIOS

Purpose: To turn on graphics text mode.

Remarks: XGTON\$ works just as GTON\$ to turn on graphics text mode. Standard console output is displayed in the graphics memory instead of alpha memory.

Example: 5000 CALL AGIOS (XGTON\$)

XGTSOFF\$

Statement: XGTSOFF\$ = FNPk16\$(4,32)

Type: Graphics, Text, AGIOS

Purpose: To turn off slanted text mode.

Remarks: XGTSOFF\$ works just as GTSOFF\$ to turn off slanted graphics mode. Subsequent graphics text will be printed in upright (unslanted) fashion.

Example: 5000 CALL AGIOS (XGTOFF\$)

XGTSON\$

Statement: XGTSON\$ = FNPk16\$(4,31)

Type: Graphics, Text, AGIOS

Purpose: To turn on slanted graphics text mode.

Remarks: XGTSON\$ works just as GTSON\$ to turn on slanted graphics text mode. Subsequent graphics text will be displayed slanted.

Example: 5000 CALL AGIOS (XGTSON\$)

XHOME\$

Statement: XHOME\$ = FNPk16\$(0,16)+“H”

Type: Alpha, Display, AGIOS

Purpose: To move the alpha cursor to the home position.

Remarks: XHOME\$ works just as HOME\$ to position the alpha cursor at the upper left hand corner of the display, the ‘home’ position.

Example: 5000 CALL AGIOS (XHOME\$)

XHOMEDWN\$

Statement: XHOMEDWN\$ = FNPK16\$(0,16)+"E"

Type: Alpha, Display, AGIOS

Purpose: To move the alpha cursor to the home down position.

Remarks: XHOMEDWN\$ works just as HOMEDWN\$ to move the cursor to the last active line of display memory. Note that this may not be the last line of the screen.

Example: 5000 CALL AGIOS (XHOMEDWN\$)

XIL\$

Statement: XIL\$ = FNPK16\$(0,16)+"L"

Type: Alpha, Display, AGIOS

Purpose: To insert a blank line in alpha memory.

Remarks: XIL\$ works just as IL\$ to insert a line in alpha memory at the current cursor position.

Example: 5000 CALL AGIOS (XIL\$)

XKBOFF\$

Statement: XKBOFF\$ = FNK16\$(0,16)+"c"

Type: General, AGIOS

Purpose: To disable the HP Touchscreen PC keyboard.

Remarks: XKBOFF\$ works just as KBOFF\$ to turn off the HP Touchscreen PC keyboard. This has the effect of not permitting any key presses until the keyboard is once again turned on or enabled.

Example: 5000 CALL AGIOS (XKBOFF\$)

XKBON\$

Statement: $XKBON\$ = FNPk16\$(0,16) + "b"$

Type: General, AGIOS

Purpose: To turn on the HP Touchscreen PC keyboard.

Remarks: XKBON\$ works just as KBON\$ to turn on or enable the HP Touchscreen PC keyboard.

Example: 5000 CALL AGIOS (XKBON\$)

XMEML\$

Statement: $XMEML\$ = FNPk16\$(0,16) + "l"$

Type: Alpha, Display, AGIOS

Purpose: To lock a portion of the alpha display on the screen.

Remarks: XMEML\$ works just as MEML\$ to lock a portion of alpha display memory at the top of the screen.

Example: 5000 CALL AGIOS (XMEML\$)

XMEMUL\$

Statement: $XMEMUL\$ = FNPk16\$(0,16) + "m"$

Type: Alpha, Display, AGIOS

Purpose: To unlock alpha memory.

Remarks: XMEMUL\$ works just as MEMUL\$ to unlock the alpha display memory locked with XMEML\$ or MEML\$.

Example: 5000 CALL AGIOS (XMEMUL\$)

XORGNSET\$

Statement: XORGNSET\$ = FNPk16\$(4,49)

Type: Graphics, Display, AGIOS

Purpose: To set the relocatable origin to the current pen position.

Remarks: XORGNSET\$ works just as ORGNSET\$ to set the graphics relocatable origin to the current graphics pen position.

Example: 5000 CALL AGIOS (XORGNSET\$)

XORGNSETCP\$

Statement: XORGNSETCP\$ = FNPk16\$(4,28)

Type: Graphics, Display, AGIOS

Purpose: To set the relocatable origin to the current graphics cursor position.

Remarks: XORGNSETCP\$ works just as ORGNSETCP\$ to position the relocatable graphics origin to the current graphics cursor position.

Example: 5000 CALL AGIOS (XORGNSETCP\$)

XORGNSETPP\$

Statement: XORGNSETPP\$ = FNPk16\$(4,27)

Type: Graphics, Display, AGIOS

Purpose: To set the relocatable origin to the current pen position.

Remarks: XORGNSETPP\$ works just as ORGNSETPP\$ to set the relocatable origin to the current graphics pen position.

Example: 5000 CALL AGIOS (XORGNSETPP\$)

XPENDWN\$

Statement:	XPENDWN\$ = FNPk16\$(4,43)
Type:	Graphics, Vector, AGIOS
Purpose:	To lower the graphics pen.
Remarks:	XPENDWN\$ works just as PENDWN\$ to lower the graphics pen at the current position. Any subsequent moves will result in a line being drawn.
Example:	5000 CALL AGIOS (XPENDWN\$)

XPENUP\$

Statement:	XPENUP\$ = FNPk16\$(4,39)
Type:	Graphics, Vector, AGIOS
Purpose:	To raise the graphics pen
Remarks:	XPENUP\$ works just as PENUP\$ to lift the graphics pen. Any subsequent moves will not cause a line to be drawn.
Example:	5000 CALL AGIOS (XPENUP\$)

XPOINTPLOT\$

Statement:	XPOINTPLOT\$ = FNPk16\$(4,48)
Type:	Graphics, Vector, AGIOS
Purpose:	To plot a point.
Remarks:	XPOINTPLOT\$ works just as POINTPLOT\$ to plot a single point at the current pen position.
Example:	5000 CALL AGIOS (XPOINTPLOT\$)

XPOLYGON\$

Statement:	XPOLYGON\$ = FNPk16\$(4,50)
Type:	Graphics, Polygon, AGIOS
Purpose:	To start polygon area fill definition.
Remarks:	<p>XPOLYGON\$ works just as POLYGON\$ to initiate the definition of a polygonal area operation. The boundary pen will be lowered.</p> <p>After using this call, continue on to specify the (X,Y) coordinates that define the polygon. You can do this by printing the FNMOVE\$ function or by setting a buffer using FNXMOVE\$. Batch commands can also be used if desired.</p>
Example:	5000 CALL AGIOS (XPOLYGON\$)

XRBOFF\$

Statement:	XRBOFF\$ = FNPk16\$(4,10)
Type:	Graphics, Vector, AGIOS
Purpose:	To turn off rubber band line mode.
Remarks:	XRBOFF\$ works just as RBOFF\$ to turn off rubber band line mode. It does not otherwise affect graphics memory or whether it is displayed.
Example:	5000 CALL AGIOS (XRBOFF\$)

XRBLON\$

Statement:	XRBLON\$ = FNPk16\$(4,9)
Type:	Graphics, Vector, AGIOS
Purpose:	To turn on rubber band line mode.
Remarks:	XRBLON\$ works just as RBLON\$ to turn on rubber band line and turn on the graphics cursor.
Example:	5000 CALL AGIOS (XRBLON\$)

XTAB\$

Statement:	XTAB\$ = FNPk16\$(0,16) + "I"
Type:	Alpha. Display, AGIOS
Purpose:	To move the cursor to the next defined tab stop.
Remarks:	XTAB\$ works just as TAB\$ to advance the alpha cursor to the next tab stop, or to the beginning of the next line if no tab stops exist to the right of the alpha cursor.
Example:	5000 CALL AGIOS (XTAB\$)

3

Practical Programming

Introduction

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

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

- User-definable Softkeys
- Keyboard Control
- Character Enhancements and Screen Control
- Touch Screen Programming

In **BPL**, we have attempted to identify the most frequently used escape sequences for your use. However, **BPL** does not define all of the escape sequences which work on the HP Touchscreen PC. For additional information, be sure to refer to the *HP Touchscreen HP Terminal User's Guide* or other HP Touchscreen PC reference guides.

Before going into details of using escape sequences, let's take a brief look at a philosophy of programming the HP Touchscreen PC.

A Philosophy of Programming the HP Touchscreen PC

The HP Touchscreen PC is both a full function personal computer and a complete intelligent terminal. Because of this dual capability, there are several ways to control the HP Touchscreen PC display and keyboard. The two methods supported by Hewlett-Packard are:

- Escape sequence programming
- AGIOS function calls

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

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 be happy with escape sequences.

The second method, called 'AGIOS', is unique to the HP Touchscreen PC. It usually requires an assembly language call to perform, and does not use standard input and output statements. You might wonder why you would want to use AGIOS if it requires such special set-up. The answer is system dependent speed: AGIOS allows console output to run as fast as 4000 characters a second!

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

You will not find an 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 and revisions of computer firmware (ROM) and operating system. If the manufacturer should change some small detail in the computer, your application may not run and your customers will be calling you.

Introduction to Escape Sequences

Escape sequences offer one of the easiest ways to access the features of the HP Touchscreen. Once you have mastered how to use these sequences, you will be able to program nearly all the features of the system: the trick is becoming comfortable with how it's done.

Imagine for a moment a printer terminal attached to a large computer. This printer is an old one, and it can only advance the paper: it cannot 'back up' to print a previous line. For the computer to print a form, it has to start at the top line and print each line sequentially. The printer terminal requires very little 'intelligence'. All it needs to do is accept a character from the computer and print it on the paper.

Some of the possible characters the printer might receive really don't 'print' anything: that is, they leave no visible mark on the paper. A 'carriage return' character is an example of such an 'invisible' letter. However, the carriage return does affect the printer: it advances the platten one line. Another invisible character is the 'line feed' character, which advances the paper to the next line and often moves the print head to the left margin.

When cathode-ray tubes (CRTs) became cost effective, the printer terminal described above was often replaced by a CRT unit. The first of these were usually replacements for printer terminals: there was still no ability to return to a previous line. As the technology advanced, however, these glass terminals became more and more advanced.

Soon, there were 'smart' CRT's which could receive special instructions from the host computer. One of the most exciting of these new features was the ability to move the print head ('cursor') anywhere on the page. This permitted an entirely new way of using computer terminals.

To accomplish this 'random' cursor positioning, the CRT needed to be sent commands from the large computer. While different types of terminals use different commands, most terminals use the same ASCII character to signify the start of a command to the terminal. This character is the 'escape' character, the ASCII character represented by the value 27 decimal.

As a smart terminal receives characters from the large computer, it 'examines' each character as that character is received. If the character is not the 'escape' character, the terminal simply 'prints' that character on the display at the current cursor location and proceeds. If the terminal receives the 'escape' character, it 'knows' the next character(s) are not to be displayed normally: they are a command to the CRT.

For our purposes here, we can treat the HP Touchscreen PC *as if* the terminal and the computer were physically separate. To the terminal part of the system, a character coming from MS-DOS is no different than a character which originated at a distant computer.

As mentioned above, the terminal examines each character it receives. When the 'computer' is the local MS-DOS system, those characters are coming from 'standard console output'. In BASIC, this means from your PRINT statements. When the 'escape' character is sent, a special part of the terminal takes control. This part is called the 'escape sequence processor'. It reads additional characters (after the escape character) and determines what action is to be taken.

The HP Touchscreen PC supports many commands, or 'escape sequences'. Some are simple and require no variable information. For example, deleting a line is simple and needs no additional information ('parameters'). The line where the cursor is located is deleted, and lines below it 'scroll up' to fill the

3-4 Practical Programming

empty space. Such simple sequences usually require only one character following the escape character to specify the command. Deleting a line occurs when an 'escape' is followed by the letter 'M'.

More advanced escape sequences are needed when additional information is required. For example, to position the cursor anywhere on the screen, you need to specify the row and column positions you want. On the HP Touchscreen, as with all other HP terminals, a special character following the escape character indicates to the escape sequence processor that a sequence is more than two characters long.

In fact, on the HP Touchscreen PC, there are four sets of extended commands indicated by the following starting characters:

- ESC & : General Alphanumeric Commands
- ESC * : General Graphics Commands
- ESC) : Character Set Control
- ESC - : Touchscreen Control

Many of these more complex commands have several parameters. To move the cursor, you specify a row and column. To define a softkey, you specify a key number, the label, the actual softkey definition, and several other parameters. The actual order in which you provide the parameters isn't critical because each has a letter associated with that parameter.

For example, one way to move the cursor is to follow the row number with the letter 'r' and the column number with the letter 'c'. The last letter in the escape sequence *must* be upper case and those before *must* be lower case. But the order is not critical. The following two sequences perform the same function, locating the cursor at the upper left corner of the screen:

- ESC & a 0 y 0 C
- ESC & a 0 c 0 Y

Further, if you didn't want to change the row number, you could specify:

ESC & a 0 C

This would cause the cursor to be positioned at the left margin on the current line. The function FNLOCATE\$ in **BPL** provides direct cursor addressing.

Summary: Using Escape Sequences

The **BPL** provides you with the most useful escape sequences. There *are* other sequences, though, so remember these guidelines when you use them:

- With two-character escape sequences, the second character may be upper or lower case. However, upper case characters have different functions than lower case characters. Be sure the case is correct depending on the function you want to use.
- Multi-character escape sequences are interpreted until the first upper case character is found after the escape character. The order of the parameters is not important.
- Send escape sequences to the console with standard console output. In BASIC, this usually means with 'PRINT' statements.



Introduction to Softkeys

There are two distinct sets of softkeys on the HP Touchscreen PC. 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 `[CTRL] [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 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".
- Return a keycode for each key, not the buffer associated with the "User-definable Softkeys".
- Are displayed by the user by typing `[Shift] [User System]`.

For more information on Application Softkeys, refer to the appropriate system documentation. Use of these keys is not covered in depth within **BPL**.

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>
```

The most commonly used form of this escape sequence is found in the **BPL** function FNSKD\$. As is true with all escape sequences, the HP Touchscreen 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 for an explanation of the various attributes.
<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.

3-8 Practical Programming



The above parameters are the ones commonly used in softkey definitions. See the *HP Touchscreen PC Terminal User's Guide* for additional softkey parameters.

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

Normal and Transmit keys are both sent to standard console input. The principal difference is that with 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 standard subroutines provided with **BPL** use INPUT\$.

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



In actual use, User-defined Softkeys may be affected by the state of the 'G' and 'H' terminal straps. When the HP Touchscreen is in local computer mode, you will generally want to set both straps to their 'open' position using the following escape sequence:

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

This is discussed in greater depth in various system manuals. You will usually find it best to set these straps at the beginning of any program using softkeys.

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-Mode" 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 **BPL** variable for each of these parameters is:

- @ = SKOFF\$
- A = SKM\$
- B = SKON\$
- S = SKL\$
- R = SKUL\$

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 modes softkey level a user would see by pressing `User System` and then `f4` if it contains the word "Modes".
- 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 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 Touchscreen PC 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. If this is not acceptable, you can define all eight User-Defined Softkeys as ‘local’ keys (attribute = ‘1a’). Further, set the buffer contents of each key to a null (empty) string. This will cause the terminal to effectively ignore any touches in the softkey area.

As mentioned above, remember the state of the ‘G’ and ‘H’ straps may affect the action of this method.

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.

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 Return 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 Touchscreen PC 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  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 effects 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. Within **BPL**, this would be a very easy sub-routine to add!

```
5000 A$ = INPUT$ (1)           'Get character without echo
5010 A = ASC (A$) : KEY = 0    'Get value; assume no KEY
5020 IF A < 18 OR A > 25 THEN 5000 'Not in softkey range
5030 KEY = A - 17              'Put KEY in range 1-8
5040 'Continue with program
```

You should find User-Defined Softkeys can be used to your advantage to make your applications easier to use.

Finally, remember the state of the 'G' and 'H' straps can affect the operation of your system unless properly initialized. If you forget to do so, your system will 'freeze' when you press the softkey the second time!

Controlling the Keyboard

Some applications on the HP Touchscreen 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 system documentation 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 Touchscreen 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 the CHRIN subroutine in **BPL** for console input. An example of using this routine is illustrated in the example below.

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

```
5005 GOSUB 3000           'Read one character w/o echo
5010 IF CH$ <> ESC$ THEN 5040 'Not ESC char
5015 GOSUB 3000           'Read the next character
5020 IF CH$ = "K" THEN 5030 'Clear Line
5025 GO TO 5005
5030 PRINT ESC$;"K";
5035 GO TO 5005           'Go get another character
5040 PRINT CH$;           'Have to echo it so it is seen!
5050 GOTO 5005
```

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

How to Get Transmit Function Key Mode Started

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

```
ESC & s 1 A Enable Transmit Mode
```

```
ESC & s 0 A Disable Transmit Mode
```

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

```
5000 PRINT CHR$(27);"& s1A";
```

Programming Considerations

Using 'transmit function keys' mode allows you to have additional control over the HP Touchscreen PC. When using this mode, there are some things of which to be aware.

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

The 'transmit function keys' mode causes the terminal to send an escape sequence to your application. The sequence you receive is the same sequence which, if sent by your application, would perform the same task as the key. To effectively control the keyboard then, your application should perform input without echo. By doing this, your application will receive the two character escape sequence and be able to determine the suitable

course of action without the HP Touchscreen having already performed the task. Finally, remember the user-defined Softkeys have default values which return two character escape sequences. 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 define 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.

One final note: in transmit function key mode, Select generates ESC&P. Your application can detect this sequence as well.

Controlling the Display

The HP Touchscreen PC 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 Touchscreen. You can use them one at a time, or in any combination of the four. Within **BPL**, the function to perform such enhancements is FNENHON\$(A\$). The value of A\$ is determined using the table below.

The general format of the escape 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 Touchscreen is different from many other systems.

Character Set Selection

The HP Touchscreen includes a number of different character 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.

The escape sequence for selecting an alternate character set is:

ESC) <cset> (FNALTCHR\$ in BPL)

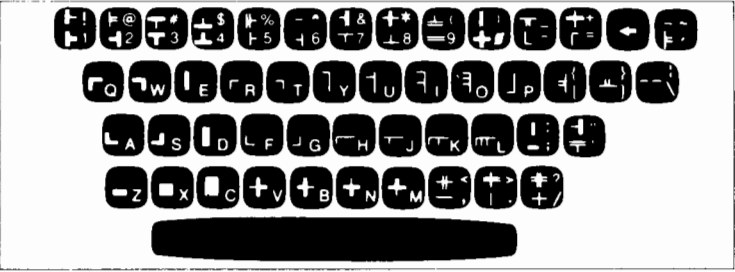
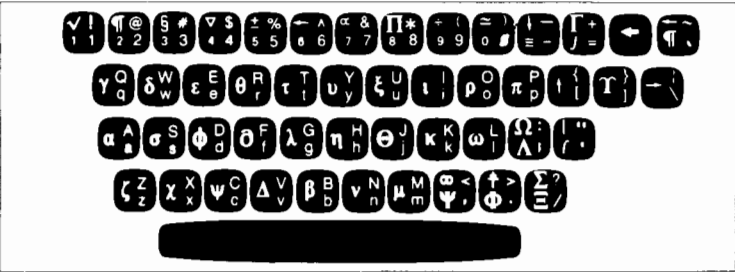
The value for <cset>, the character set selection, is:

- @ : 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) characters. The decimal values for these characters is 14 and 15 respectively. The BPL variables for these characters are ALTCHROFF\$ and ALTCHRON\$ respectfully.

You alternately use the alternate set by simply printing the ASCII character which corresponds to the desired character:



You can write programs 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 rectangle on the screen:

```
5000 'Print a box with 'Hello There' in the middle
5010 PRINT CHR$ (27);" B"; 'Select Line Drawing as Alternate
5020 PRINT CHR$ (14);" Q;;;;;;;;;;;;;;;;;;;;;;;;;W" 'Top line
5025 ' Now print middle line with normal characters in middle
5030 PRINT CHR$ (14);".";CHR$ (15);" Hello There ";
5040 PRINT CHR$ (14);"." 'Other end of box
5045 ' That concludes the middle line
5050 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 5025 through 5045).

Note that within **BPL** you can replace the awkward 'CHR\$' functions with the defined variables 'ALTCHRON\$' for 'alternate character set on; and 'ALTCHROFF\$' for 'alternate character set off'.

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. There is also 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 (BPL ADON\$)
ESC * d F	Alpha Memory Off (including softkey labels) (BPL ADOFF\$)

Programming Considerations

In using both display enhancements and alternate character sets, keep in mind how the HP Touchscreen PC 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.

Using Touch Screen

One of the most unique features of the HP Touchscreen PC 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 Touchscreen PC information about where you want to detect touches, and the system firmware does 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! Use **BPL** functions FNTF\$ or FNTOUCH\$.

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	<p>This parameter specifies whether the alpha cursor should be positioned at the upper left corner of the field. Possible values for < curs > are:</p> <p>0 = The cursor does not move to the field. 1 = The cursor is positioned at the upper left corner of the field.</p> <p>If "p" is not specified, the cursor is not positioned on touch.</p>
< beep > b	<p>The "b" parameter specifies whether the system should beep when the field is touched. Valid values for < beep > are:</p> <p>0 = No sound occurs when the field is touched. 1 = The system beeps on touch.</p> <p>If "b" is not specified, no beeping occurs when touched.</p>
< on__enh > e	<p>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 the table below.</p> <p>If "e" is not specified, the default on-enhancement of 10 is used. This displays a half-bright inverse field.</p>
< off__enh > f	<p>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.</p> <p>If "f" is not specified, the default off-enhancement of 2 is used. This causes an inverse video enhancement in the field.</p>

Touch Enhancements Table:

<parm> value:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Blk:		*	*		*	*		*	*		*	*		*	*	
Inv:			*	*			*	*			*	*			*	*
Ulin:					*	*	*	*					*	*	*	*
HBrt:									*	*	*	*	*	*	*	*

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

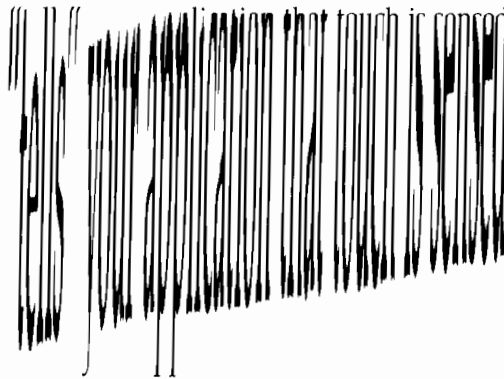
- 1 = ASCII field
- 2 = keycode field
- 3 = toggle field
- 4 = normal field

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 Touchscreen



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 Touchscreen PC 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.
- 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

Here, <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

Here, <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 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 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.

The most common uses of these touch modes are in **BPL** variables TSON\$ and TSOFF\$

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
```

Use **BPL** variables TSDDEL\$ and FNTFDEL\$ to perform these functions.

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. Use **BPL** variable **TSRESET\$** to perform a Touchscreen reset.

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).

Within **BPL**, the functions which are used to define fields use ASCII fields exclusively. You will probably find these the easiest to use with your application.

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

5000 PRINT CLS\$;	'Clear screen w/ cursor at top
5010 GOSUB 7000	'Define a touch field: actual ' field not critical
5020 GOSUB 3000	'Read one character from touch 'using CHRIN routine
5030 IF CH\$= " 1" GOTO 5500	'Go off on option 1 to unshown line
5040 GOTO 5010	'Not option 1, return for more

Now look at Program B:

PROGRAM B

5000 PRINT CLS\$;	'Clear screen w/ cursor at top
5010 GOSUB 7000	'Define a touch field: actual ' field not critical
5020 GOSUB 3000	'Read one character from touch 'using CHRIN routine
5030 IF CH\$= " 1" GOTO 5500	'Go off on option 1
5040 GOTO 5020	'Not option 1, return for more

See the difference? Look at Line 5040. 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

```
5000 PRINT CLS$;           'Clear screen w/ cursor at top
5010 GOSUB 6000             'Define a touch field
5015 PRINT CHR$(27);"-z2N"; 'Turn on sensing
5020 GOSUB 3000             'Read one character from touch
                             'using CHRIN routine
5025 PRINT CHR$(27);"-z0N"; 'Turn off sensing
5030 IF CH$="1" GOTO 5500   'Go off on option 1
5040 GOTO 1015             'Not option 1, return for more
...
6000 PRINT CHR$(27);"-zg1,5r1,5c1b1a2m1L1";
6005 RETURN
```

Note lines 5015 and 5025. They turn sensing on just before input is expected, and turn it off immediately after input is received.

Of course, at a line later in the program such as line 5500 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 5020. To avoid inputting additional (unwanted) characters.

If more than one field is defined over a particular area of the screen, the HP Touchscreen PC 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.

Device Control From BASIC

For both the BASIC compiler and BASIC interpreter there exists a problem concerning device control. Although a program can OPEN a device as a file (eg: COM1), the BASIC code assumes it to be some sort of disc drive and has difficulty communicating with it. This is not a bug, BASIC was simply not designed to do device control.

Two problems exist when using standard BASIC statements (OPEN, INPUT, PRINT) to try to control devices. The first problem is that of buffering. BASIC will wait for either a record separator or buffer over-flow before continuing execution after a read. In many cases, neither of these will take place. As far as writing to a device goes, you must force a buffer over-flow or close the file in order to force the write to complete. The second problem concerns MS-DOS. In its default file access mode, MS-DOS performs a lot of clean-up when reading information from a device (strip escape characters, look for a `CTRL C`, etc.). This is called "cooked" mode. The extra time needed to "cook" the data usually causes data-loss on high-speed devices.

The device control functions in this package are designed to eliminate the above problems. First, they do not use fixed length buffers. As an example, if you write 25 bytes, they will be sent out immediately. If you read 37 bytes, the function will wait for 37 bytes to become available, then read them. You could even request an I/O CONTROL read, which would try to read the 37 bytes but would not wait if 37 weren't available. It would read as many as it could and return them. The problem concerning "cooked" mode is also eliminated. Before any read or write, the file is temporarily modified into the "raw" mode state ("uncooked").

There are three functions contained in the DFUNCS files:

DOPEN

This function will open a device file and return a unique file number associated with the file. This file number must be used to identify the file when calling the following routines.

DACCESS

This function allows read, write, or I/O CONTROL read access to an open file.

DCLOSE

This function allows you to close (discontinue use of) a file.

This package contains copies of the functions for both the interpreter and the compiler, their related source listings (assembly language), and test programs designed to give you more information on how to call the procedures. The interpreter copy, DFUNCSIB.IMG, must be BLOADED into memory before the functions can be called. If you are using the compiler, you can simply LINK the DFUNCSCB.OBJ file in with your code.

You will find additional documentation on these device control functions on your BPL Master Disc. The file called 'DEVICE.DOC' can be printed by a word processor or simply COPY'ed to your printer using File Manager or MS-DOS Commands. Study that file and the program examples provided to master device file control from within BASIC.

One final note: the sample programs contained with BPL to illustrate device control have been tested with the HP 7470 plotter and an HP 3000 mainframe. If you wish to use a different plotter, or to connect to a different mainframe, some modification may be required.

Using AGIOS from BASIC

As you have read, AGIOS calls are slightly more difficult to use than escape sequence programming. However, the benefits include faster execution and better control over the HP Touchscreen PC.

Rather than document these more advanced calls and complicate this manual, we have included a file on your BPL Master Disc which describes AGIOS usage in detail. The file, called 'AGIOS.DOC', can be printed using a word processor, or COPY'ed to your printer using File Manager or MS-DOS Commands. To use AGIOS, print that file and study its examples and text.

4

Variables and Functions by Group

Basic Programmer's Library Variable & Function Listing by use

Function	Description	Has Comparable AGIOS Function	Page Number Reference
AGIOS			
FNPK16\$(MSB,LSB)	FuNction PaCK 16 bit		2-27
FNSOFF\$(X1)	FuNction String OFFset		2-30
FNSOFFL\$(X1)	FuNction String OFFset Long		2-30
FNXBAT\$	FuNction AGIOS BATch		2-34
FNXTCESC\$	FuNction AGIOS Execute ESCape Sequence		2-40
Alpha			
Display control			
ADOFF\$	Alphanumeric Display OFF	✓	2-1
ADON\$	Alphanumeric Display ON	✓	2-1
ALTCHROFF\$	ALternate CHaRacter set OFF		2-2
ALTCHRON\$	ALternate CHaRacter set ON		2-2
CLR\$	CLear		2-7
CLS\$	CLear Screen		2-7
COFF\$	Cursor OFF	✓	2-7
CON\$	Cursor ON	✓	2-8
ENHOFF\$	ENHancement OFF\$	✓	2-9
FNALTCHR\$(BUF\$)	FuNction ALternate CHaRacter set		2-10
FNCENTER\$(BUF\$,W)	FuNction CENTER		2-13
FNENHON\$(BUF\$)	FuNction ENHancement ON	✓	2-18
FNLOCATE\$(R1,C1)	FuNction LOCATE	✓	2-25
HOME\$	HOME	✓	2-46
HOMEDWN\$	HOME DOWN	✓	2-47
MEML\$	MEMory Locked	✓	2-48
MEMUL\$	MEMory UnLocked	✓	2-49

Function	Description	Has Comparable AGIOS Function	Page Number Reference
Alpha Editing			
BS\$	Back Space		2-6
BTAB\$	Back TAB	✓	2-6
CLL\$	CLear Line	✓	2-6
CR\$	Carriage Return	✓	2-8
DL\$	Delete Line	✓	2-8
FNUP\$(BUF\$)	FuNction UP		2-23
IL\$	Insert Line	✓	2-47
TAB\$	TAB	✓	2-57
Alpha Softkey Control			
FNSKD\$(K,L\$,RET\$)	FuNction SoftKey Definition		2-28
SKL\$	SoftKeys Locked		2-55
SKM\$	SoftKey Modes		2-55
SKOFF\$	SofKeys OFF		2-56
SKON\$	SoftKeys ON		2-56
SKUL\$	SoftKeys UnLocked		2-56
General			
BELL\$	BELL		2-5
ESC\$	ESCaPe		2-9
KBOFF\$	KeyBoard OFF	✓	2-47
KBON\$	KeyBoard ON	✓	2-48
Graphics Display Control			
FNDRAWMODE\$(X1)	FuNction set DRAW MODE	✓	2-15
FNGCMA\$(X1,Y1)	FuNction Graphics Cursor Move Absolute	✓	2-20
FNGCMR\$(X1,Y1)	FuNction Graphics Cursor Move Relative	✓	2-20
GCLS\$	Graphics CLear Screen	✓	2-40
GCOFF\$	Graphics Cursor OFF	✓	2-40
GCON\$	Graphics Cursor ON	✓	2-41
GDFT\$	Graphics DeFaulTs		2-41
GDOFF\$	Graphics Display OFF	✓	2-42
GDON\$	Graphics Display ON	✓	2-42
GPDFT\$	Graphics Picture DeFaulTs	✓	2-43

4-2 Variables and Functions by Group

Function	Description	Has Comparable AGIOS Function	Page Number Reference
GRESET\$	Graphics RESET	✓	2-43
GSET\$	Graphics SET	✓	2-44
GSETUP\$	Graphics SET UP		2-44
Graphics			
Origins			
ORGNSET\$	ORiGiN SET	✓	2-49
ORGNSETCP\$	ORiGiN SET Cursor Position	✓	2-49
ORGNSETPP\$	ORiGiN SET Pen Position	✓	2-50
Graphics			
Polygons			
BDRYPENOFF\$	BounDaRY PEN OFF	✓	2-3
BDRYPENON\$	BounDaRY PEN ON	✓	2-4
FNBOXFA\$(X1,Y1,X2,Y2)	FuNction BOX Filled Absolute		2-11
FNBOXFR\$(X1,Y1,X2,Y2)	FuNction Box Filled Relative		2-12
FNDEFPAT\$(BUF\$)	FuNction DEFine area fill PATtern		2-14
FNFILLPAT\$(X1)	FuNction set FILL PATtern	✓	2-19
POLYGON\$	POLYGON	✓	2-52
Graphics			
Text			
FNGTLABEL\$(BUF\$)	FuNction Graphics Text LABEL		2-21
FNGTORGN\$(X1)	FuNction Graphics Text ORiGiN	✓	2-22
FNGTROT\$(X1)	FuNction Graphics Text ROTate	✓	2-23
FNGTSIZE\$(X1)	FuNction Graphics Text SIZE	✓	2-24
GTOFF\$	Graphics Text OFF	✓	2-45
GTON\$	Graphics Text ON	✓	2-46
GTSOFF\$	Graphics Text Slant OFF	✓	2-46
GTSO\$	Graphics Text Slant ON	✓	2-46
Graphics			
Vectors			
FNBOX\$(X1,Y1,X2,Y2)	FuNction BOX		2-12
FNDEFLINE\$(X1,X2)	FuNction DEFine LINE pattern		2-13
FNDRAW\$(X1,Y1)	FuNction DRAW	✓	2-15
FNLINEPAT\$(X1)	FuNction set LINE PATtern	✓	2-24
FNLYNE\$(X1,Y1,X2,Y2)	FuNction LYNE (line)		2-25
FNMOVE\$(X1,Y1)	FuNction MOVE pen	✓	2-26

Function	Description	Has Comparable AGIOS Function	Page Number Reference
FNPSET\$(X1,Y1)	FuNction Point SET		2-28
PCMOVE\$	Pen to Cursor MOVE		2-50
PCPLOT\$	Pen to Cursor PLOT		2-50
PENDWN\$	PEN DoWN	✓	2-51
PENUP\$	PEN UP	✓	2-51
POINTPLOT\$	POINT PLOT	✓	2-51
RBLOFF\$	Rubber Band Line OFF	✓	2-54
RBLON\$	Rubber Band Line ON	✓	2-54

Internal Strings

BUF\$	BUFFer
DR\$	DRive
DS\$	Data Segment
FILE\$	FILE
L\$	Label
RET\$	RETurn
REV\$	REVision
REV.BY\$	REVision BY
REV.DATE\$	REVision DATE

Internal Numeric

AGIOS	Alpha Graphics Input Output Subsystem
C1	Column or Column 1
C2	Column 2
DS%	Data Segment
GETCH	GET CHAracter
GETDS	GET Data Segment
K	Key
LSB	Least Significant Bit
MSB	Most Significant Bit
R1	Row or Row 1
R2	Row 2
TLA	Three Letter Acronym
W	Width
X1	X1
X2	X2
Y1	Y1
Y2	Y2

4-4 Variables and Functions by Group

Function	Description	Has Comparable AGIOS Function	Page Number Reference
Touch Screen			
FNTF\$(R1,R2,C1,C2,RET\$)	FuNction Touch Field		2-31
FNTFDEL\$(R1,C1)	FuNction Touch Field DELeTe		2-31
FNTOUCH\$(R1,C1,RET\$)	FuNction TOUCH define		2-32
TSDEL\$	Touch Screen DELeTe		2-57
TSOFF\$	Touch Screen OFF		2-57
TSON\$	Touch Screen ON		2-58
TSRESET\$	Touch Screen RESET		2-58

Subroutines

String

CH\$	CH\$		
------	------	--	--

Subroutines

Numeric

CH%	CH%		
CI%	CI%		
SKFLAG	Soft Key FLAG		

5

Alphabetic Listing of All BPL Variables and Functions

This chapter cross-references the variables and functions of **BPL**. For a subject index refer to the contents of 'INDEX.DOC' found on your **BPL** master disc. This file can be printed using either your word processor or the MS-DOS COPY command.

ADOFF\$
ADON\$
AGIOS
ALTCHROFF\$
ALTCHRON\$
BDRYPENOFF\$
BDRYPENON\$
BELL\$
BS\$
BTAB\$
BUF\$
C1
C2
CH\$
CH%
CI\$
CLL\$
CLR\$
CLS\$
COFF\$
CON\$
CR\$
DL\$
DR\$
DS\$
DS%

ENHOFF\$
ESC\$
FILE\$
FNALTCHR\$
FNBOX\$
FNBOXFA\$
FNBOXFR\$
FNCENTER\$
FNDEFLINE\$
FNDEFPAT\$
FNDRAW\$
FNDRAWMODE\$
FNENHON\$
FNFILLPAT\$
FNGCMA\$
FNGCMR\$
FNGTLABEL\$
FNGTORGN\$
FNGTROT\$
FNGTSIZE\$
FNLINEPAT\$
FNLOCATE\$
FNLYNE\$
FNMOVE\$
FNPK16\$
FNPSET\$
FNSKD\$
FNSOFF\$
FNSOFFL\$
FNTF\$
FNTFDEL\$

FNTOUCH\$
FNUP\$
FNXBAT\$
FNXDRAW\$
FNXDRAWMODE\$
FNXENHON\$
FNXFILLPAT\$
FNXGCMA\$
FNXGCMR\$

5-2 Alphabetic Listing of All BPL Variables and Functions

FNXGTORGN\$
FNXGTROT\$
FNXGTSIZE\$
FNXLINEPAT\$
FNXLOCATE\$
FNXMOVE\$
FNXTCESC\$
GCLS\$
GCOFF\$
GCON\$
GDFT\$
GDOFF\$
GDON\$
GETCH
GETDS
GPDFT\$
GRESET\$
GSET\$
GSETUP\$
GTOFF\$
GTON\$
GTSOFF\$
GTSON\$
HOMES\$
HOMEDWN\$
IL\$
K
KBOFF\$
KBON\$
L\$
LSB
MEML\$
MEMUL\$
MSB
ORGNSET\$
ORGNSETCP\$
ORGNSETPP\$
PCMOVE\$
PCPLOT\$
PENDWN\$
PENUP\$

POINTPLOT\$
POLYGON\$
R1
R2
RBLOFF\$
RBLON\$
RET\$
REV\$
REV.BY\$
REV.DATES\$
SKFLAG
SKL\$
SKM\$
SKOFF\$
SKON\$
SKUL\$
TAB\$
TLA
TSDDEL\$
TSOFF\$
TSOON\$
TSRESET\$
W
X1
X2
Y1
Y2
XADOFF\$
XADON\$
XBDRYPENOFF\$
XBDRYPENON\$
XBTAB\$
XCLL\$
XCLR\$
XCOFF\$
XCON\$
XDL\$
XENHOFF\$
XGCLS\$
XGCOFF\$

5-4 Alphabetic Listing of All BPL Variables and Functions

XGCON\$
XGDOFF\$
XGDON\$
XGPDFT\$
XGRESET\$
XGSET\$
XGTOFF\$
XGTON\$
XGTSOFF\$
XGTSON\$
XHOMES\$
XHOMEDWN\$
XIL\$
XKBOFF\$
XKBON\$

