Ultra Rom Board/Editor
APB—102 Owner's Manual

6842 Valjean Avenue
Van Nuys, California 91406
(213) 989-1204

WARRANTY INFORMATION

Hollywood Hardware Inc. warrants the products it manufactures against defects in material and workmanship for a period of one hundred twenty (120) days from the date of purchase. During the warranty period Hollywood Hardware Inc. will repair or replace at no charge any product which becomes defective. The defective product must be returned to Hollywood Hardware Inc., or an authorized repair center, with a description of the problem and dated proof of purchase, such as a Bill of Sale, cancelled check, etc.

This warranty does not apply if in the decision of Hollywood Hardware Inc. the product was damaged as a result of accident, misuse, or misapplication, or by unauthorized service or modification of the product.

No other warranties are expressed or implied, including, but not limited to, the implied warranties of merchantablility or fitness for a particular purpose. Hollywood Hardware Inc. is not responsible for incidental and/or consequential damages incurred by the use or misuse of any product. The determinaton of applicability of this product for any particular purpose is solely the responsibility of the purchaser. This warranty gives you specific legal rights which vary from state to state. Some states do not allow the exclusion or limitation of incidental or consequential damages. so the above limitation or exclusion may not apply to you.

COPYRIGHT INFORMATION

Global Program Line Editor (GPLE) is copyrighted 1983 by Neil Konzen and is sold under license from Synergistic Software, acting as his agent.

All other programs in the Ultra Rom Board, including but not limited to the Operating System of the Ultra Rom Board and the Ampersand Programming Utilities are copyrighted 1983 by Hollywood Hardware Inc.

"Apple" and "Applesoft" are registered trademarks of Apple Computer Inc.

APB-102 ULTRA ROM BOARD OWNER'S MANUAL

TABLE OF CONTENTS Page

ULTRA ROM BOARD

General Description:

The Ultra Rom Board provides 32K bytes of additional memory space for the Apple][and Apple][e computer systems. It will accept up to eight 2716 or 2732 EPROMs (Erasable, Programmable Read Only Memories), without hardware modification. By itself this would be of value only to experienced programmers, but it comes with a complete Global Program Line Editor and a variety of Ampersand (&) Utilities which make it a powerful Program Development System for beginners right up through the most experienced high-level and machine-language programmers. With GPLE and the Ampersand Extensions, Applesoft Basic becomes a serious programming language, and program development time can be reduced by a substantial factor. And since only 8K bytes of the rom board are currently used, 24K bytes remain for future expansion. This provides space for a number of additional programming aids currently in the works at Hollywood Hardware, as well as room for your own utilities, if you have assembly language knowledge and the ability to program EPROMs.

Who the heck is HOLLYWOOD HARDWARE?

Hollywood Hardware was formed by a group of technical people who have been providing special effects to the film industry for years. Among our credits are "STAR WARS", "STAR TREK, THE MOTION PICTURE", "FIREFOX", "BATTLESTAR GALACTICA", "CHINA SYNDROME", and many more. In creating our special effects we use Apple][computers extensively. Among other things, we have designed, built, and programmed a multichannel motion control system using an Apple for model photography. The Apple controls up to 16 axes of movement of the camera, models, and puppets, and all in real time in sync with the frame rate. In the process of doing this, we have developed many unique peripheral boards for the Apple which we are now providing to the general public. Since we built these products for ourselves first, we designed them to be durable. These systems have done service all over the world, from the Arctic wastes of Greenland to the dry heat of the Arizona desert, and they have to work right all the time.

We feel that we design a better product because we're not just producers of Apple peripherals, we're users of Apple peripherals as well.

What is GPLE?

GPLE is a powerful Global Program Line Editor. In Rom it becomes a part of your Apple][or Apple][e computer system just like Applesoft itself. GPLE provides an extremely powerful set of editing functions, like the most powerful word processors, for editing your Basic programs. It includes INSERT, DELETE, ZAP, RESTORE, FIND, and more. The word "Global" means that GPLE can be told to change a variable or phrase everywhere it occurs in a program.

In addition it includes a variety of built-in Macros. The word "Macro" is computer jargon for the process of redefining a single key to output a series of keystrokes to take the drudgery out of commonly executed commands. The Macros on the Ultra Rom Board are initiated by hitting the ESC key, followed by the key for the Macro you desire. In addition to the many built-in Macros, you have the ability to define your own Macros which can be kept on disk, so that you can customize GPLE to suit your own needs.

Why GPLE on rom?

The original GPLE, as written by Neil Konzen, resided in high memory, thereby reducing the available memory space by about 4K bytes. The powerful features of this editor made this sacrifice worthwhile, but some very large programs would not run with it in the Apple. This required reloading GPLE every time one wanted to re-edit the program under construction.

Later versions of GPLE were moved to the 16K ram card (if one was available) and this solved the problem for many applications, but still not all. Here at Hollywood Hardware we had just such a problem. We had written and were using an extremely large program consisting of Basic and machine-language modules for realtime control of robotics, and we were constantly rebooting GPLE to work on our program. In order to avoid the time and hassle of this constant disk access, we conceived of the Ultra Rom Board. Now these editing features are never more than 4 keystrokes away, and if GPLE does conflict with the way a program functions (an infrequent occurrence) it can go away just as quickly.

NOTES ABOUT SYNTAX

To avoid confusion we will use the following symbols and their meanings throughout this Manual.

1) Control characters will be enclosed in square brackets. For example, [A] = Control A and [SHIFT M] = Control Shift M. Control characters are typed by holding the CTRL key down WHILE typing the key in brackets. In the case of [SHIFT M] and [@], the CTRL key AND the SHIFT key must be held down while the appropriate key is pressed. PLEASE NOTE: on the][e the [SHIFT M] is actually []].

2) One special Control key is [M]. This is the character produced by the key marked RETURN on your keyboard. To make the function of this Control character clearer, we will use the symbol <cr> (short for Carriage Return) to stand for this key.

3) In general, whenever we are describing somthing you would type at the keyboard, we will enclose it in slashes. For example: /YOU TYPE THIS/. Do not type the slashes, just what's inside.

4) Escape keys differ from Control keys in that you type the ESC key FOLLOWED by the Macro key. The words "Escape functions" and "Macros" will be used interchangably in this Manual.

5) In examples of Escape sequences (e.g. /ESC H/), the space between the "ESC" and the "H" are for

clarity only. In this example the "H" should be the first key typed after the "ESC" key is hit.

INSTALLING THE ULTRA ROM BOARD

The Ultra Rom Board can be installed in any slot except slot 0. Normally, slot 1 is reserved for a printer (Pascal, for example, will accept a printer only here), slot 3 is an 80 column card, and slot 6 is the disk controller. Slots 2 or 7 are good choices, but any free slot will work fine. In order to install the Ultra Rom Board you MUST first turn off the Apple][. In fact, if you forget to turn off the Apple before plugging in any peripheral card, you probably won't be playing with your Apple again until it gets back from the repair shop. So to be on the safe side, why not unplug the Apple while you are working on it?

Once the Apple is off, remove the top cover. This is done by grasping the rear edge to the cover and pulling up firmly until the fastners separate (or come off altogether in many older Apples). The slots in the Apple][are numbered from 0 to 7 from left to right as seen from the front. The slots in the Apple][e are numbered from 1 to 7. In both cases, the slot numbers are printed on the circuit board above the connectors. Carefully line up the edge connector on the Ultra Rom Board with the peripheral connector of your choice (it will go in only one way), and firmly push down until it seats.

That's all there is to it! Now replace the cover, put the Hollywood Hardware Demonstration disk in drive 1, and turn the Apple back on. In the next section we will guide you through your first session with the powerful features of your new Ultra Rom Board.

FIRST SESSION

This section is for people who are unfamiliar with GPLE. If you feel that you do not need this tutorial, jump ahead to the section, "USING THE MACRO TABLES".

NOTE:

> While going through this tutorial, if you find that you do not understand something, or you do not get the results that you expect, take a minute and review "NOTES ABOUT SYNTAX" on page 4. Whenever we want you to type something, we will enclose it in slashes like this: /you type this/. Do not type the slashes, just what's inside. Also, so that you won't confuse the key marked "RETURN" with the Basic command of the same name, whenever we are refering to the key, we will use the symbol "<cr>" (short for CARRIAGE RETURN).

Now, enter the following frivolous program into the Apple without using GPLE.

```
10   REM STAR WEAVER
20   HOME : VTAB 24
30   FOR X = 1 TO RND (1) * 40 : H = INT (RND (1)
     * 39) + 1 : HTAB H : PRINT "*"; : NEXT :
     PRINT
40   IF PEEK (-16384) < 128 THEN 30
50   POKE -16368,0
60   HW = 0 : VT = 12.1 : VB = 12 : FOR HL = 19 TO
     0 STEP -1 : HW = HW + 2 : VT = VT - .6 : VB =
     VB + .6 : POKE 32,HL : POKE 33,HW : POKE
     34,VT : POKE 35,VB : HOME : NEXT
```

FIRST SESSION

This is a simple program which weaves a random pattern of stars on the screen and then erases it when any key is pressed. Make sure that the program is correct by running it and fixing any mistakes the old way. Now it is time to invoke GPLE. Simply type /PR#n/, where "n" is the number of the slot the Ultra Rom Board is in. You should see "(C) 1983 HOLLYWOOD HARDWARE" at the top of the screen and the flashing cursor at the bottom of the screen. The screen may or may not have been cleared (we'll explain this later). This tells you that GPLE is now active and that the Ampersand (&) functions have been hooked up. That's all there is to it.

Now type /ESC O/. You have just used your first Macro. It's a bit faster than typing "HOME" and it will work in Integer basic too. Next type /ESC L/. You have to hit "<cr>" after this one because it's waiting for you to add the line number(s) you want to LIST, if any. Just hit /<cr>/ and you'll see that your program is still there. GPLE can be connected and disconnected as often as you like with the Ultra Rom Board, without affecting the program in memory. Now try typing /[Q]/. That's how easy it is to disconnect GPLE. Your Apple is now just like it was before GPLE was hooked up, except that the Ampersand utilities are still available. Go ahead and turn GPLE back on (remember: /PR#n/).

Now let's try to edit the program using GPLE. Let's say that we wanted to make line 60 a subroutine, so that we could use it instead of "HOME" to clear the screen in a more interesting way. Without GPLE it would be necessary to retype the whole line just to add a RETURN at the end. Try the following instead. First, type /[E]/. As soon as you hit the "[E]" the word "EDIT" should appear on the screen. If not, hit /<cr>/, and then hit /[E]/ again. The "[E]" must be the first character typed after a "<cr>" to enter the EDIT mode.

Now that the word EDIT is on the screen, GPLE is waiting for you to tell it what to edit. In the section titled, "EDITING WITH GPLE", all of the ways to use the editor are fully described, but right now we just want to edit line 60. So type /60/ and hit /<cr>/. Just like that, line 60 appears on the screen with the cursor positioned at the first character of the line. Try typing /[N]/. The cursor has moved to the eNd of the line. Now type /[B]/ and you'll find yourself back at the Beginning of the line. Next type /[F]/. If you have an Apple][, the cursor will stop blinking. This means that GPLE is waiting for something to Find. Type /:/ and see what happens. Notice that the cursor is still not blinking. Type /:/ again. You can move through a line very quickly this way, Finding characters of interest. Any key other than ":" now will get you out of the Find mode, and WILL BE ENTERED INTO YOUR TEXT. This is so that you can Find characters and easily change them. To avoid changing your line, hit the right arrow key /-->/.

So far, we've seen some ways to move around in the edit mode, let's examine some ways of changing the line. Type a /[D]/. You have just Deleted one character at the cursor. Now type a /[D]/ and hold down the /REPT/ key at the same time (or just wait a second if you have an Apple][e). This is an easy way to Delete a bunch of characters. There is a more convenient way to Delete a lot of characters, however, and it's called Zap. Get back to the Beginning of the line (/[B]/) and then hit /[Z]/. The cursor stopped blinking again, didn't it? It's waiting for the character you want to Zap to. Type some character contained in what's left of your line and see what happens. Zap stays in the Zap mode just like Find stays in the Find mode. So hit the space bar (/ /), and we'll go on.

Well, your line 60 must be pretty messed up by now. You certainly wouldn't want to change it to this! Not to worry. Type /[R]/. You have just Restored your line 60. You can now continue to edit it without fear, because you can always Restore it. You could also have typed "[X]" which would have eXited the edit mode without changing line 60.

Back to our goal of changing this line to a subroutine. Type /[N]/ to get to the eNd of the line and add /:RETURN/. Now, just to make a point, type /[B]/ to get back to the Beginning and then type /<cr>/. Notice that you do not have to move the cursor to the end of the line to enter it using GPLE, a considerable time savings. Type /ESC L 60 <cr>/ to see that line 60 has been changed to a subroutine.

Now a couple more changes are necessary to make our program work with a subroutine. Using what you have learned, edit line 10 to read:

```
10   GOSUB 60 : VTAB 24 : REM STAR WEAVER
```

and add lines 45 and 50 as shown:

```
45   GET I$ : IF I$ = "E" THEN GOSUB 60 : END
50   GOTO 20
```

Try running this program. Notice how easily we were able to add these features to the program. Before we leave this tutorial and turn you loose on your own programs, let's try using a couple of Macros.

There are two different Macro tables available when the Ultra Rom Board is connected. They are called the Rom Macro table and the Ram Macro table. You have already used Rom Macros when you used /ESC L/ to LIST a program and /ESC O/ to clear the screen. The Rom Macro table is on the Rom board itself, and cannot be changed (except by "blowing" another EPROM). The Ram Macro table, however, is available for you to create, edit, and save to disk. In this way you can build custom Macro libraries of functions YOU find most useful. If you redefine a key that's already in the Rom Macro table, your definition will have priority until you reinitialize the Rom board, or turn the computer off.

Let's try making a new Ram Macro. First, type /[E]/, and when the "EDIT" prompt appears type /ESC G/. Notice that the letter "G" has appeared on the screen, with the cursor to the right of it. GPLE is ready to add the Macro "G" to the Ram Macro table. If "G" had already been defined, the whole Macro would now be displayed for editing, unless it was a Rom Macro. Remember that Rom Macros cannot be edited, so GPLE does not even try.

After the letter "G" type /GOSUB 60/. Hit /<cr>/ and then try typing /ESC G/. As you can see, the command "GOSUB 60" has appeared on the screen just as if you had typed it at the keyboard. Now hit <cr>, and you get your result. So let's edit this Macro. Type /[E] ESC G/. Then type /[I]/. You are now in the Insert mode, so insert a /'/ (single quote). Go to the end of the line (remember: [N]) where we want to add a <cr>, but wait! If you hit the <cr> right now, you will just enter Macro as it is. This is where [O] (for Overide) comes in. Type /[O]/. Now the next Control key you type will be inserted into the text. So now hit the /<cr>/. The inverse "M" is the symbol for [M], which is the character that the <cr> key produces. Whenever you are editing, Control characters will show as inverse.

Now, type /<cr>/ again to enter your new Macro. Before we try it out, type /[E]/ again and this time type a /?/ after "EDIT ". Hit /<cr>/ and you will see the first page of the Macro table. Any key will make it continue listing, and the space bar will stop and single step through the list. Try it! When you get to the end of the list you will see your new Ram Macro, separated by a blank line from the Rom Macros. This is how to list the Macro tables.

It's time to try the changes to your Macro, so type /ESC G/. This time the screen is cleared without ever seeing the command that did it (this is what the tick ¼'¶ did) and without having to hit <cr> (because it was embedded in the Macro itself). There is one more interesting Macro to try before we leave, so let's edit line 10 once more.

Hit /[E] 10 <cr>/. Then move the cursor just past "VTAB 24" and delete the rest of the line with [D]'s. Now hit /ESC [X]/. You are in the insert mode, so just type /STAR WEAVER <cr>/. List your program again. Notice how nicely your REMark stands out this way? The "ESC [X]" can be typed from anywhere in the line because it always goes to the end of the line.

There is much more to learn about the capabilities of GPLE and the Ampersand functions, but the best way to become familiar with them is to play with a simple program such as this. The next sections go into more detail about the operation of the Ultra Rom Board, GPLE, and the Ampersand Utilities. Try everything you read about and your speed at writing programs will soon be many times faster than it is now.

USING THE MACRO TABLES

All Macros (also called Escape Functions) begin by hitting the ESC key. The Macro corresponding to the next key typed is then executed. A Macro is simply a series of keystrokes which have been assigned to one key. When you use a Macro, it is as if you were typing the entire sequence from the keyboard. Macros can be used when you are entering informaton at the keyboard or while editing. For example, instead of typing "CATALOG,D1", just type /ESC 1/. Instead of typing /LIST/, just hit /ESC L/. And instead of typing "LOAD MYPROGRAM, D1", just type /ESC 1/, use /I/ to move the cursor to the line with MYPROGRAM on it, and hit /[L]/. Not only is it faster, but you won't misspell MYPOGRAM either.

There are actually two Macro tables active whenever the Ultra Rom Board is hooked up. The Rom Macro table is on the Rom Board, and cannot be changed. If you type the sequence /[E] ? <cr>/ you will see a listing of the available Rom Macros. There is also a Ram Macro table which can be changed by the user at will. This Macro table is initially located in Page 3 of Ram memory (see "MEMORY MAP"). It can be moved by the user to anywhere in Ram Memory, to create really enormous Macro tables. It can also be saved to disk for later use. This is covered in greater detail later.

Macros can also be nested. This means that one Macro can call another. Whenever GPLE encounters a [SHIFT M] ([]] on the][e) in a Macro definition, it executes the next character as a Macro. Some Macros are repetitive, which means that they remain in the ESCAPE mode after they have finished doing whatever it is they do. An example of this is the cursor control cluster, I,J,K and M. After moving the cursor, the next key struck is also interpreted as a Macro. If the last character in a Macro definiton is the [SHIFT M] then it is repetitive. On Apple]['s, the cursor will stop blinking whenever you are in the ESCAPE mode. The Apple][e has a different cursor style.

As we said before, when you use a Macro, it is as if you were typing the characters at the keyboard. For example, look at the following Escape definition:

```
MPRINT"I AM A NICE COMPUTER"<cr>
```

In this definition, M is the Macro key and PRINT"I AM A NICE COMPUTER"<cr> is the sequence of characters which would be issued when ESC M is typed. The <cr> at the end of this Macro is used to symbolize a [M], or RETURN, embedded in the definition. When you put a <cr> in a Macro, it will execute as soon as you hit the Macro key. Otherwise it will wait for you to hit the RETURN key. When executed, this Macro would print out:

```
]PRINT"I AM A NICE COMPUTER"
]I AM A NICE COMPUTER
```

It looks just like it would if you had typed it in from the keyboard, doesn't it? If you wanted this Macro to execute without printing the instruction that caused it, you could use the symbol ('), which is called the tick. Whenever a tick is encountered in a Macro definition, the remainder of the Macro will not be printed to the screen, but will be executed. This means that if the Macro above was changed to read:

```
M'PRINT"I AM A NICE COMPUTER"<cr>
```

it would print out:

```
]I AM A NICE COMPUTER
```

whenever "ESC M" was typed. Exactly what is going to happen when you start nesting Escape functions that include ticks can get a little tricky. The best approach is to experiment with different combinations until you feel comfortable with this very powerful feature.

Several utilities have been provided to help you develop and use your own Macros. Macros can be added to the table or edited using GPLE (see "Editing the Macro table"). A small amount of space is available all the time for short Macros in Page 3. For example, if you were working on a program which changed Zero Page address $6, and you wanted to see the value of that address at a certain point in your program, you might create the following Macro:

```
HPRINT PEEK(6)<cr>
```

In this way you could run your program, stop it at some point, and just type "ESC H" to see the value of G. This is the sort of Macro that you create once and get rid of as soon as you find the bug in your program.

You can create much more space for Macros, however, if you want to build more extensive Macro tables. One simple way to get more space is to use one of the Ampersand utilities called &MOVMAC. This utility when typed at the keyboard will create a new Macro area 256 bytes below HIMEM, and reset HIMEM to protect it. In this way you can have really extensive Macros.

For example, you might have a Macro that you use to initialize your printer which looks like this: PPR#1 <cr> [I]80N [I]60P [I]I [I]M [I]F [X]. When you typed /ESC P/ this Macro would turn on a printer in slot 1, initialize it to print 80 columns, skip over perforations, keep the screen on, maintain these parameters after the printer is deselected, and format the Basic listing (if you have the same printer card that I do).

To save the entire Ram Macro table to disk all you have to do is to LET M$ = a short, descriptive name and then type /ESC [SHIFT M]/. Your Macro table will be saved to the currently active disk with the names "MACROTBL.YOURNAME" and "MACROPTR.YOURNAME". For example, if you LET M$="PRINTER", your Macro table would be saved to disk as "MACROTBL.PRINTER" and "MACROPTR.PRINTER". To reload a Macro table, just BLOAD both files back into memory.

Another special Macro will tell you where your Ram Macro table currently is. Type /ESC =/ to see the address and space allotted to your Macro table. When you BLOAD a Macro table from disk, it is a good idea to use this Macro to find out where it is and then set HIMEM: to protect it. There is a program on the Demonstration Disk which shows how to do this in a program.

If you run out of space in a Macro table you will hear a beep when you try to enter a Macro, and if you look at it again you will find that you have lost some characters at the end. You can make a file bigger by using the following procedure.

1) Save your Macro table to disk using the technique described above.

2) Type /&MOVMAC/ to create a larger area for Macros.

3) Type /ESC =/. This will tell you the current address of the Macro table area.

4) Type /BLOAD MACROTBL.yourname,A<address from step 3>/

You can now continue to add to this Macro table until you fill it up again. To see a complete list of available Macros type /[E] ? <cr>/. This listing can be stopped and started using the normal listing control keys. If you want more information about how the Macro tables are arranged, see "APPENDIX A - ADVANCED TOPICS."

There are too many Macros for us to go into detail about each of them here, but one set of Macros is particularly useful, and perhaps not obvious, so we should give an example its usage. These are the DOS Macros. We mentioned that whenever you are in the ESCape mode, the cursor stops blinking. Try the following example on your computer.

First, hit /ESC 1/ and <cr>'s as necessary to get to the end of the catalog of disk 1. Now, type /ESC/, and the cursor stops blinking (unless you have a][e). Now, use the /I/ key to get up to an Applesoft program you would like to load into the Apple. When the cursor is on the same line as the program (the cursor is still not blinking, right?), type /[L]/. Suddenly, the word "LOAD" appears, and the cursor is all the way at the right of he screen, waiting for you to hit <cr>. If you do not want to load this program, simply type /[X]/ to cancel the instruction. You can LOAD, SAVE, RUN, BLOAD, BSAVE, BRUN, LOCK, UNLOCK, and DELETE disk files using built-in Macros, and never misspell a program name again. If you want to BLOAD, you use the same sequence, except that you type /[B]/ then /[L]/.

EDITING WITH GPLE

There are three categories of text which can be edited using GPLE:

1) Basic program lines
2) Macro table definitions
3) Commands being typed at the keyboard

The Editor can be entered in one of two ways. If you wish to edit Basic program lines or Macro table definitions, simply type a [E] as the first character after a <cr>. When the prompt "EDIT " appears on the screen the line number or Macro to be edited can be entered. The legal syntax for the editing command is described below. Entering the editor while typing commands at the keyboard is initiated in a different manner. At any time while entering text from the keyboard (e.g. typing a DOS command, adding a line to a Basic program, etc.), you can type a [W] and everything that you have typed up to the cursor will be displayed on the screen and can be edited using the normal editing keys. In other words, if you routinely type [W] before adding lines to a Basic program, you will effectively always be in the edit mode.

Editing Basic lines:

Type [E] and then the line or range of lines you wish to edit. Some examples:

EDIT 60	edit line 60
EDIT ,60	edit all lines from the beginning of the program

	through line 60
EDIT 60,	edit all lines from 60 through the end of the program
EDIT 60,90	edit all lines from 60 through 90

You can also edit only program lines which contain a certain string. Some examples:

EDIT "ABC"	edit all lines containing the string "ABC"
EDIT "AB?"	edit all lines containing a three character string starting with "AB". In other words, the "?" can be used as a wild card character
EDIT "ABC","DEF"	edit all lines containing "ABC" and change all occurrences of "ABC" to "DEF". GPLE stops at each line to allow you to accept the change.

When you are using this mode to search and replace, GPLE stops at each line where a match is found. You can either accept the change by hitting /<cr>/, restore the line and continue searching by hitting /[R] <cr>/, or abort the processs by hitting /[X]/. You can also make any other changes to each line before continuing. If you do not want GPLE to stop at each line you can add a "/F" (for Fast search) to the end of the edit command, as shown:

```
EDIT "ABC","DEF"/F
```

It should be noted that GPLE does not parse the search string, so the spacing of the commands must be exactly as they appear in the listing for the string to be found. GPLE will also Pack a long line (to remove spaces) if necessary in order to edit it, and this could cause it not to find a search string.

It should also be noted that GPLE will not find a string embedded in a longer string. It expects the string it is looking for to be delimited by a space, quote, parenthesis, colon, semicoln, Basic command or some other character which is clearly not part of the string. If you want to find strings embedded in other strings you must append "/R" (for Raw search) to the end of the edit command. The Raw search is somewhat slower because all possible matches must be checked. Here is an example of the syntax for a Raw search:

```
EDIT "ABC","DEF"/R
```

The various editing parameters can be used in any combination as long as the commas are in the right location. The only exception is that you cannot have a Raw search and a Fast search active at the same time. The following is a perfectly legal edit search command:

```
EDIT 60,,"ab?DE","FGH"/R
```

There is one other form the editing command can take where Basic lines are concerned, and that is:

`EDIT .`	This will re-edit the last edited line.

NOTES ABOUT EDITING BASIC PROGRAM LINES

1) GPLE will automatically Pack long lines to remove spaces if necessary in order to edit it. This can affect Global searches. In addition, it may be unable to edit some very long lines at all. For this reason, if you are going to Crunch a program for code effeciency, you should do all of your editing on the unCrunched version.

2) Escape functions can be used while editing. In fact, editing keys can be embedded in a Macro very effectively to enhance editing. For example, a Macro which reads, "H[I]FOR W = 1 TO 50: NEXT :" could be used to insert a short delay into a line with only two keystrokes. If you wanted to add this instruction to several lines, this Macro would be very handy indeed. You have to be careful, however, not to use a Macro which has a <cr> in it, as this will change your line and exit the edit mode, making the change official.

3) Whatever your line looks like on the screen when you hit <cr> will be entered to the Apple. This means that if you are editing line 10 and you change the line number to 66, when you hit <cr> you will have a new line 66 which is exactly like line 10. This is an easy way to move lines around in your program. (Don't forget that this is not a true renumbering scheme because it doesn't change any other line that might reference the line you move, although you could use the global search and replace to fix all such occurrences)

4) The technique described in note 3 can also be used

to test how a line will work. If you edit line 10, and delete the line number altogether, the line will execute when you hit <cr> just as if you had typed the whole thing in. A simple Macro to do this for you might be: /T[B][Z]<space><cr>/.

5) When the "EDIT " prompt is on the screen, you are already in the EDIT mode, so you can search for control characters and edit your parameter line using the normal editing keys.

EDITING THE MACRO TABLE

The Ram Macro table can be edited by typing /[E] ESC/ followed by the character to be edited or added to the Ram Macro table. Because the built-in Macro table is in ROM (Read Only Memory) it cannot be edited, and so GPLE does not even try. Any key that you define overides the built-in function, so you must be careful not to redefine keys (like [F] and +) which are used by other important Macros. If you start getting funny results from old familiar Macros, this is probably what happened.

Ram Macros are edited exactly like Basic program lines with the exception that the [R] key does not work. It is a good idea to pack each Macro definition ([P]) to remove unnecessary spaces before entering it into the table. This conserves Ram memory space. Macros can be removed from the macro table either by typing [@], which totally clears the Macro table, or by editing the Macro and typing [Q] immediately.

There are a few items of special importance which should be noted about editing Macros:

1) Control characters, which can be embedded in Macro definitions by using [O], will be displayed as inverse characters.

2) If the last character in an Escape function is a [SHIFT M] (which shows up as an inverse "]" and is in fact a []] on the Apple][e) then the Macro will be repetitive. The next key typed after this Macro executes will also be interpreted as a Macro.

3) If a [SHIFT M] is inserted into a Macro, then the next character in the line will be interpreted as a Macro. In other words, one Macro can call another.

4) If a tick (') is included in a Macro, the rest of the line will not be printed (unless a printer is turned on) and only the output or the effect of the Macro will be seen.

THE EDITING KEYS

[B]	Moves the cursor the beginning of the line.
[C]	Converts the character at the cursor to the opposite case and advances the cursor. This key and the repeat key is one way to convert print statements to lower case for Apples without that capability. If your Apple does not have lower case display then these characters will look like gibberish, but a printer will display them properly.
[D]	Deletes the character at the cursor and moves the rest of the line in.
[F]	Finds the first occurrence of the next letter typed. This command is repetitive, and will stay in the Find mode until you type a different character or until it cannot find another match. The first character you type to exit the Find mode is entered into the text, to allow you to easily find and replace characters. If this is undesirable you should leave the Find mode with one of the arrow keys.
[I]	Inserts characters at the cursor. Remains in effect until one of the arrow keys, an edit key, or other control key is hit.
[M]	This is the same as a Carriage Return. It exits the Edit mode and accepts the whole line as it looks on the screen.

[N]	Moves the cursor to the eNd of the line.
[O]	Overides normal input, inserts a control character at the cursor, and enters the insert mode. The [O] must preceed each control character to be inserted. This is a very useful function which allows you to put Carriage Returns, and left and right arrows into PRINT and REM statements for much more control of your printing and listing format. One Macro (ESC [X]), for example, uses this approach to add a REM statement and four Carriage Returns to the end of a program line. This allows you to insert nicely spaced remark statements into a Basic program without a lot of hassle.
[P]	Packs the text. This removes all spaces from the text, which may be necessary in very long lines to keep from exceeding Apple's limit of 239 characters on a line. If you start getting inverse "G"'s while adding to a line you are editing, these are the beeps you would be getting if you were typing in a long line normally. GPLE will automatically pack a long line if it is necessary in order to edit it. Be careful with long lines as you may lose some of the end of your line, even if you Pack it.
[Q]	Quits the editing mode and accepts everything up to the cursor. This is what you are used to on the Apple when you hit return; you lose everything

after the cursor.

[R] Restores the line to its condition before you started editing it. You can use this command as many times as you like if you make mistakes. NOTE: this command does NOT work while editing Macros or if you entered the editor with a [W].

[X] EXits the editor, but leaves the line unchanged. A RESET while in the edit mode will have the same effect.

[Z] Zaps all characters up to the next key pressed. In other words if the first key pressed after the [Z] is a ":", then all characters up to the first colon will be deleted, just as if that many [D]'s had been typed. Zap is repetitive, like Find. To exit the Zap mode type any key other than the key you are Zapping to.

THE AMPERSAND PROGRAMMING UTILITIES (APU)

An Overview:

The second Rom on the Ultra Rom Board is the APU. This Rom contains a collection of utility programs and Applesoft extensions as well as the operating system necessary to coexist with other Ampersand utilities in Ram or other Roms on the Ultra Rom Board. These programs are intended to extend the capabilities of Applesoft and/or to help the Basic programmer use hidden features of the Apple more easily and to interface to machine language programs. Before we take a detailed look at the various Ampersand functions available, a brief description of what the Ampersand command is and how it works is in order.

The Ampersand is the "&" symbol found above the "6" on Apple]['s and above the "7" on the Apple][e. This symbol is actually an Applesoft command, just like RUN, PRINT or GOTO. When Applesoft encounters this symbol from the keyboard or in a program line it immediately jumps to a certain location in Ram memory (see: "Memory Map"). By changing this location in memory to point to the Ultra Rom Board, it is possible for us to take control of the computer temporarily to perform some useful function. And because Applesoft jumps to this program before even checking to see what comes after the "&", we can create whole new commands that would ordinarily cause SYNTAX ERRORs. This is how all Ampersand routines work.

This simple Applesoft command has proven surprisingly useful and has been used in a wide range of imaginative utilities. We have written a collection of routines of our own, some of which duplicate functions found in published listings or software offerings, and have included them in our APU Rom.

THE AMPERSAND PROGRAMMING UTILITIES (APU)

Each utility is used like any other Applesoft command. For example, to find the current setting of HIMEM, you simply need to type /&HIM<cr>/ and the answer will be displayed. Or if you want to turn on the high resolution graphics page 1, but without erasing what you had previously drawn, you could type /&HGR/ to do just that.

Not all Ampersand functions can be used in programs. Many are intended to help you while you are writing a program, and have no relevance in a running program. If you try to include such a command in a program, you will get an ILLEGAL RUNTIME ERROR when the program encounters it. This error will always halt program execution, whether or not an ONERR statement has been issued. The legal runtime commands, of course, can be used at any time.

The APU is hooked up every time the Ultra Rom Board is selected by a PR#n. It is not, however, disconnected when the board is turned off with a [Q]. This is to allow you to continue to use the Ampersand utilities even if GPLE has a conflict with your program. It should be noted that the "&" is an Applesoft command. There is no equivalent command in Integer Basic, and the use of the Ampersand in Integer Basic will only cause a SYNTAX ERROR.

THE AMPERSAND COMMANDS

The following is a description of all of the Ampersand Utilities in the APU rom:

&

Mode: Immediate

The Ampersand with no parameters will display a catalog of utilities currently available in the Ultra Rom Board. If the proper protocol is followed, user written utilities can also be included in the catalog (see "Putting your own programs on the Ultra Rom Board"). To exit the catalog at any point, use ESC or E.

&[P]xxxxx

Mode: Runtime

The ampersand followed by [P] will pass the command "xxxxx" to whatever utility was connected at $3F5.$3F7 (1013.1016) when the Ultra Rom Board was first hooked up. This allows other Ampersand utilities to be used with no conflict with APU.

Note: to change all of the Ampersand commands in a program to this format, use the edit line:

```
EDIT "&","&[P]"/F.
```

&ASC

Mode: Runtime

Prompts for a keypress and then prints the ASCII equivalent of that key, with the hi bit set and cleared. This can be useful when you are reading the keyboard directly (using PEEK(-16384)), and you want to look for a specific key, or when you need the value of a specific control character for a CHR$(n) declaration.

&BIN

Mode: Immediate

Prints the address and length of the most recently BLOADed file. **ASSUMES APPLE STANDARD DOS**.

&CEOL

Mode: Runtime

Clear to end of line. This duplicates CALL -868, but is easier to remember.

&CEOP

Mode: Runtime

Clear to end of page. This duplicates CALL -958, but is easier to remember.

&DISCHR

Mode: Immediate

Displays all Control characters as inverse. This is useful for examining disk files, program lines, or program output for embedded Control characters. This command remains active until you EDIT something, reinitialize the Ultra Rom Card, hook up a printer, or hit RESET.

Note: This is a short routine that lives in the input buffer, so a very long input line will destroy part of this code and may cause the system to hang. If so, a RESET will rehook everything.

&FINDLINE<decimal number>

Mode: Immediate

Find the absolute memory address of any line in an Applesoft program. Can be used to find the end of a program by adding a temporary line at the end and then finding the address of it.

&FRESEC

Mode: Immediate

Prints the number of free sectors remaining on the last disk accessed. This utility assumes a standard DOS.

&HIM

Mode: Immediate

Prints the current setting of HIMEM.

&HGR

Mode: Runtime

Identical to Applesoft command HGR, except that it does not clear the screen. This is useful if you want to go to the TEXT mode for instructions or user input and be able to return to graphing without erasing what has been done.

&HGR1

Mode: Runtime

Displays the full high resolution graphics screen, like &HGR2, but using page 1. Does not clear the screen.

&HGR2

Mode: Runtime

Identical to HGR2, but without clearing the screen. If used with &HGR1, this can be useful for quickly switching between two graphics pages for special effects.

&IF <exp> THEN <instruction(s)> : &ELSE <instruction(s)>

Mode: Runtime

This is an extenstion of the IF <exp> THEN <instruction(s)> command in Applesoft. If the expression after &IF is true (non zero) then the instruction(s) following the THEN but before the &ELSE will be executed. If the expression is not true (zero), then the instruction(s) after the &ELSE will be executed. &IF/THEN/&ELSE statements can be nested, and other Ampersand commands can be used within them, but the statements must be completed on a single line.

&LOM

Mode: Immediate

Prints the current setting of LOMEM.

&MOVMAC

Mode: Immediate

Moves, or rather reinitializes, the Ram Macro table from GPLE to a location 256 bytes below HIMEM, and moves HIMEM down to protect it. If the Ram Macro table
has already been moved, it is moved down another 256 bytes. This can be done repeatedly to create very large Macro tables. If you have a Macro table on disk that you would like to move into this space, simply type /ESC =/ and then /BLOAD MACROTBL.yourname,A address given/. You can then build on to your Macro table in this way.

Note: because of some housekeeping space required, you actually get only 253 bytes for your first Macro table.

&NOTNEW

Mode: Immediate

Restores a program in memory which has been lost through an accidental NEW, FP, or INT. &NOTNEW should be used immediately after the old program was erased. If any variables have been defined or a SYNTAX ERROR has occured since the program was lost, then this utility will attempt to salvage the program, and will warn you with the message "PROGRAM DAMAGED" that all is not well. Typically, some of the bytes in the first line (or lines, if very short lines are used) will have been overwritten with zeros, and this utility will replace those bytes with question marks, and set that line number to 0. If you have a lot of work invested in this version of the program, it may be worth it to save this damaged copy in a TEMPORARY file, and use an earlier version to reconstruct the first line(s). Programs shorter than 3 lines cannot be found.

&PRCMD<expression>

Mode: Runtime

Prints the character or Applesoft command equivalent to the value of <expression>. This utility is a subroutine for a future formatted lister program, but can be useful for examining memory, or perhaps writing you own formatted lister. An example program which uses this command is included on the Demonstration Disk.

&PRINT USNG:<string>;<variable(,variable)>

Mode: Runtime

Uses the string after the colon as a template for printing the variable after the semicolon. The string can be a literal (e.g. "Cost = $.00"), variable (e.g. STRING$), or an array variable (e.g. STRING$(1,5)). The variable can be a literal (e.g. 123.456), real (e.g. COST), integer (e.g. COST%), or real or integer array. If a decimal point is present, then the output will be rounded to the number of decimal places designated in the string. The number is always RIGHT justfied in the string, and "0"'s, "*"'s, "$"'s and spaces are always replaced by a number, if there is one for that position. Any other character in the string will not be replaced. If there are too many decimal places in the number to fit in the string, only "#"'s will be printed to indicate an overflow (rather than risk printing an incorrectly truncated value). The dollar sign, if present, will "float" to the left of the number if possible. There is a program on the Demonstration Disk which has examples of &PRINT USNG statements.

&RESMAC

Mode: Immediate

Resets the Ram Macro table to its default Page 3 location. Assumes that the Macro table has been moved using &MOVMAC. Unlike &MOVMAC, this command cannot be used more than once in succession. In order to protect DOS, this routine will not change HIMEM if the Macro table is already in Page 3.

&SM"<string>",Rr,Ss

Mode: Immediate

Searches all memory for the string in quotes. The high bit is ignored for this search. This is useful for finding the absolute address of a particular string in a program, or to find a command in DOS or Applesoft, etc. The search starts at $D000 (53248) and wraps around through zero to end at $BFFF (49151). The R and S designations are optional, but if they are specified then Bank r of the ramcard in Slot s is searched in the $D000 - $FFFF memory space. If S is not specified then the ramcard is assumed to be in slot 0. The values for r must be in the range 0 - $F. Only hexadecimal numbers are allowed and the $ is optional. The values for s must be in the range 0 - 7. See "Memory Map" for an explanation of the Bank designaions for 64K and 128K Ram cards.

Note: do not specify R or S if you are using Applesoft in the Ram card, as the search will naturally take place there. If you are using the Big Mac Assembler or a similar program which uses the language card, and your Ram card is larger than 16K, then this search might leave the wrong bank selected, and your next attempt to use the program might fail. If so, search again using R0.

&SM"$<hexadecimal string>",Rr,Ss

Mode: Immediate

This is similar to the above, except that it searches for a match for a hexadecimal string. Spaces between numbers are allowed, but are not required. If an odd number of bytes are listed, then the last byte is assumed to be single. The R and S parameters function exactly as described above. This utility is useful for finding subroutines and data in machine language programs (e.g. &SM"$20 ED FD" would find all of the locations in memory where a JSR $FDED was used).

&SP<string>

Mode: Immediate

This utility searches the Applesoft program for a match to <string> and prints the line numbers where a match occured. Applesoft commands can be included in the string, and spaces do not matter, except within quotes, because the line is parsed exactly like an Applesoft line before the search begins. This search is basically equivalent to GPLE's Raw search (e.g. EDIT "ABC"/R), with four differences:

1) As stated above, spaces do not matter unless they are between quotes.
2) Only line numbers are printed.
3) This search is much faster, which can be substantial in long programs.
4) The output of this search (as with all Ampersand functons) can be sent to a printer.

&SP"<string>

Mode: Immediate

This functions exactly like the above except that the string is not parsed before the search begins. This is required if you are searching for a string which may contain an Applesoft command, such as "friENDly". No trailing quote is allowed, unless it is to be included in the search.

OPERATION OF THE ULTRA ROM BOARD

The Ultra Rom Board is turned on by typing /PR#n/ where "n" is the number of the slot the rom board is in. Once the board is invoked, it is not affected by RESET or commands such as INT, FP, MAXFILES, NEW, CLEAR, etc. When the rom board is turned on it hooks itself into the stream of characters coming from the keyboard, and going to the screen. In this way it can check for command characters and it can control the output of listings and Macros, etc. In addition, it saves the address that the Ampersand vector ($3F6,$3F7) is pointing to and hooks itself up instead. When Ampersand commands are issued that aren't in the Ampersand rom, they are sent on to this address. This is explained in more detail in the section entitled, "THE AMPERSAND PROGRAMMING UTILITIES (APU)".

The Ultra Rom Board can be disconnected by typing /[Q]/ as the first character after a <cr>. When disconnected the reset vector is returned to its normal warmstart condition, and the input and output hooks are restored to normal. In all ways except one, the computer is exactly like it was before the rom board was hooked up. The one exception is the Ampersand hook. Because many of the Ampersand functions are designed to be used in running programs, the Ampersand utilities are left active when the board is turned off, and will remain active until some other utility claims the Ampersand vector. Even then, many utility programs will also pass on Ampersand commands which are not intended for them, and so our Ampersand utilities would still be active. Some experimentation is necessary to find out which is the case for any particular application.

OPERATION OF THE ULTRA ROM BOARD

The Ultra Rom Board does claim some ram memory space. The area from $300 to $3CF is used by the Operating System for the Rom board, as well as by GPLE and the APU. Historically, this Page 3 area has been used for short machine language programs, but since many of the programs which would have used this space are either already in the Ampersand rom, or can be put in rom or elsewhere in memory, this was seen as a reasonable compromise. If memory conflicts do occur in this area, the Ultra Rom Board will disconnect itself with an announcement to that effect. (An example is a program which tries to use CHAIN from Applesoft with GPLE active.) Some memory conflicts can cause the rom board to hang, in which case a RESET will cause the Ultra Rom Board to reinitialize the page 3 area.

When the Ultra Rom Board is active RESET and PR#n do not effect the page 3 area unless certain bytes have been disturbed. To completely reinitialize page 3 use [@] as the first key typed after a <cr>.

COMPATIBILITY WITH OTHER PROGRAMS

When it is first hooked up, the APU saves the address at the Ampersand vector (normally simply a jump to a return) in a safe area in Page 3. If the APU cannot find a command in its library, it will pass it on to the Ampersand routine that was in the system first. If no Ampersand routine was in the system, or if the command is not intended for that routine either, then you will get a SYNTAX ERROR.

If the Ampersand utility that you are using has commands that are the same as commands in the APU, then simply insert a [P] after the "&", and the APU will pass the command on without checking it further. The [P] will be removed (more specifically, the text pointer is advanced past it) so that the next routine will see the command as normal.

The APU uses the Zero Page addresses from $6 to $9 (6 to 9) and from $F9 to $FF (249 to 255), but it saves these addresses on entry, and restores them on exit, so that as far as any other program is concerned, no Zero Page addresses are disturbed (see "MEMORY MAP"). The one exception to this statement is when errors are encountered by Applesoft while an Ampersand utility is working (typing [C] is an error, by the way). For example, if you are using &PRINT USNG: and you get a SYNTAX ERROR, Applesoft itself stops the program and issues the error message, so the APU never has a chance to restore Zero Page. Most programs are not that sensitive about Zero Page, but you should be aware of this detail in case you start getting problems with another utility.

If Macros are not being used, or the Macro table has been moved to another location, it is possible to use the Macro table area from $35B through $378 (859 through 888) for (very) short machine language programs or variable storage.

If GPLE causes a memory conflict with a running program (e.g. an attempt to use CHAIN, which overwrites Page 3), you can turn the Rom board off by issuing /PR#0/ and /IN#0/ before executing the offending statement. The Rom board can be turned back on with the normal /PR#n/. You should follow this statement with a /PRINT/ statement if you are using it in a program. The GREETINGS program on the Demonstration Disk has an example of a program which will find and turn on the Ultra Rom Board if it is in the system.

APPENDIX A - ADVANCED TOPICS

This section is intended primarily for Assembly Language programmers who wish to develop their own EPROMS, or programs which will interface with the software on the Ultra Rom Board. Appendix C contains a map of the memory usage of GPLE and the APU, and Appendix D contains a source listing for the Driver portion of the Rom Board plus some example subroutines.

Provision has been made in the Operating System of the Ultra Rom board to allow routines in one Rom to access subroutines in another Rom, returning cleanly with only an RTS. The protocol necessary to connect to the Ampersand routines and the Ampersand catalog is described in full below. Examples of Assembly Language Macro's (based on the "Big Mac" assembler) are provided which allow two banks of a 2732 EPROM to appear ALMOST contiquous.

MEMORY USAGE

It may be helpful to refer to the Memory Map in Appendix C while reading this section.

The I/O space

All of the Roms on the Rom Board share the Memory space from $C800 through $CFFF. Indeed, they share this space with Roms on other peripheral boards as well, buffered printer interface cards being an obvious example. The Rom board can be viewed as two banks of eight 2K Roms. The "low" banks are numbered 0 - 7 (high bit cleared) and the "high" banks are numbered $8 - $F (high bit set). GPLE occupies banks 0

& 8 and APU occupies banks 1 & 9. The Driver firmware is actually the first 256 bytes of bank 0.

NOTE:

> In order for a 2716 EPROM to appear to be the low bank of a 2732, the high and low banks of the 2732 are actually swapped. When blasting a 2732 you must arrange the banks opposite of the way you wish to access them (i.e. bank 0 is in the high half of the 2732 and bank 8 is in the low half)

Zero Page

The Rom board uses several Zero Page addresses, but all of the free Zero Page addresses are saved and restored upon exit; therefore, memory conflicts with other programs are unlikely. Several other Zero Page addresses are used as pointers, but they have been chosen to avoid conflicts with the routines that share them.

Page Three

The Operating System on the Ultra Rom Board requires the unrestricted use of the Page 3 space from $300 (768) to $3CF (975). The Operating System continually checks several bytes in this area to see if the space has been overwritten by another program, and will either disconnect or reassert itself if these bytes are altered (depending on what it is doing at the time).

APPENDIX A - ADVANCED TOPICS

Scratch Pad Ram

None of the I/O scratch pad Ram addresses in the screen buffer have been used (see "APPLE][REFERENCE MANUAL", Page 82, or "APPLE][e REFERENCE MANUAL", Page 125). The user may use this space by getting the number from SLOT1, ANDing it with $7, and using this number as an index into the scratch pad Ram space

THE OPERATING SYSTEM

From a hardware standpoint, a given Rom is selected by reading the address $C0sn, where s = the slot the Rom board is in + 8, and n = the number of the bank to be selected. Under normal circumstances, however, the Operating System takes responsibility for selecting Rom banks and passing control between subroutines.

The Ultra Rom Board is initialized by the use of PR#n, IN#n, or JSR $Cn00, where "n" is the number of the slot the Rom board is in. This routine ends with an RTS, so it can be called by a running program, although it will print the "HOLLYWOOD HARDWARE" header and beep the speaker whether you like it or not. If you do not want it to clear the screen, you can VTAB 24 from Basic, or set CV to $17 before calling the Rom Board. If you are initializing the Board from Basic, you should follow the command with a PRINT statement, or your next printed character will be lost.

The byte at $300 is called AMPTEST. If this byte is $A5 then the Ampersand vector has already been set up by the Operating System, and the pointer to the Ampersand Utility which was in the system when GPLE was connected has been saved at AMPSAV ($3CA.3CB). If you want the Rom Board to reconnect the Ampersand vector, place a zero at AMPTEST, and do a /PR#n/, RESET, [R], /JSR $Cn00/, etc.

The byte at $301 is called MACTEST. It functions like AMPTEST, but as a test for the Ram Macro table. If this byte is NOT $A5 then the Ram Macro table is assumed to be garbage and is established at $35B (859). The pointer to the beginning-1 of the Macro table is located at $379.37A (889.890). The pointer to the end-2 of the Macro table is located at $37B.37C (891.892). You may establish a Ram Macro table anywhere in memory by setting up the beginning and ending pointers and placing a 0 in the first byte of the Macro table.

APPENDIX B - WRITING YOUR OWN PROGRAMS FOR ROM

The Ampersand Programming Utility Rom (APU) is set up to recognize another Rom in bank 2. For a Rom to be recognized, it must have an $A5 at $CFFE. If the characters following an "&" do not match the commands in the APU library, it will pass control to the Rom in bank 2 at $C800. Similarly, the Ampersand catalog will pass control to the Rom in bank 2 at $C803. Future Hollywood Hardware Roms will follow the same protocol of passing control to the next higher Rom bank.

Intercepting an Ampersand Command

Address $C800 in your Rom should be a JMP to the routine that evaluates the characters after the Ampersand to see if they are intended for you. On entry, TXTPTR ($B8.B9) is pointing at the first character after the "&", and the A register contains this character. The Ampersand command will always be terminated by a zero or a ":".

If the command is intended for you then, of course, you should take whatever action is appropriate. You should exit with a RTS with the A register set to 0 to notify the APU that action was taken. The APU will take care of cleaning up the TXTPTR to satisfy Applesoft.

If the command is not yours, you should restore the registers to their entry values, restore TXTPTR, if necessary, and exit with the following code (See Appendix C for the values of these labels).

```
PASSAMP     LDA     #<ROMBADR
```

```
        STA     ROMJMP+1
        LDA     #>ROMBADR
        STA     ROMJMP+2

        LDA     SLOT1       ;$Cn
        PHA
        LDA     #<GPINPUT-1 ;$3C
        PHA
        LDA     ROMBNK
        STA     NXTROM
        INC     NXTROM      ;Get next higher Rom
        JMP     $B7         ;CHRGOT - Restores ACC
```

This routine will pass control to the next Rom, if there is one, and/or to an Ampersand utility in Ram if there was one in the system when the Rom board was hooked up.

NOTE:

> The free Zero Page addresses are not saved before passing control to your Rom, and the decision as to whether or not to save and restore them is up to you. However, if you wish to use them, there are two routines in bank 1 called SAVZ and RESZ which may be called using the techniques described below in "Calling Subroutines in Another Rom"

Intercepting the & Catalog

Address $C803 should contain a JMP to your subroutine to print a catalog of choices to the screen. This routine should end with the following code (See Appendix C for the values of these labels).

```
CATALOG  LDA    #<CATBADR  ;Not necessary if no one
         STA    ROMJMP+1   ; has changed ROMJMP
         LDA    #>CATBADR  ; since coming from
         STA    ROMJMP+2   ;APU

         LDA    SLOT1      ;$Cn
         PHA
         LDA    #<GPINPUT-1 ;$3C
         PHA
         LDA    ROMBNK
         STA    NXTROM
         INC    NXTROM     ;Get next higher Rom
         RTS               ;To GPINPUT
```

This code will pass the Ampersand catalog on to the next higher Rom or return to Basic it is in the last Rom in the chain.

Calling Subroutines in Another Rom

It is possible to use subroutines in other Roms, even though they reside in the same memory space as your Rom, by using GPINPUT. GPINPUT saves the number of the currently active Rom bank on the Rom Stack. It then turns on the Rom specified by NXTROM, makes it the active Rom and JSR's to it via ROMJMP (which has been set up by the calling routine). On return it pops the last Rom bank number off the stack, turns it back on and RTS's to the calling routine. The Rom stack can be used up to 15 levels deep.

The following routine will demonstrate this technique. See Appendix C for the values of these labels.

```
JSRIND    STA     PTRO                ;Save ACC
          LDA     #<SUBROUTINE        ;Low byte
          STA     ROMJMP+1
          LDA     #>SUBROUTINE        ;High byte
          STA     ROMJMP+2
          LDA     SLOT1
          PHA
          LDA     #<GPINPUT-1
          PHA
          LDA     #ROMNUM             ;# of the Rom that
          STA     NXTROM              ; SUBROUTINE is in.
          LDA     PTRO                ;Restore ACC
          RTS
```

You must JSR to this routine as if it were the subroutine you were calling. For example, if SUBROUTINE above was COUT, then you might label JSRIND "COUTIND". To use COUT, you would simply JSR COUTIND and control would return to you when COUT in the other Rom executed an RTS. Status register flags cannot be passed in this manner, but all registers will be intact. The Macros entitled "ROMCALL" and "ROMCONT" in Appendix D give an example of two routines which work together with an address table to minimize the code required to use this technique with many subroutines.

Because many printer cards contain EPROMS which might conflict with the APU, whenever a routine is called which can output a character, APU treats it as a ROMCALL. In this way, even if a printer card turns off the APU, it will be turned on again as it returns through GPINPUT.

Bank Switching Techniques

It is possible to execute JSR's and JMP's between banks of a Rom in such a way as to create the appearence on continuity between the two banks. For this purpose we have created three Macros (using the "BIG MAC" assembler) which are called "BANKSW","JSRBS" and "JMPBS", and are listed in Appendix D. The subroutine BANKSW must be present in both banks of the Rom at $CFE3. A JSRBS then looks like this:

```
JSRBS    PHA                        ;Save ACC
         LDA    #<SUBROUTINE        ;Low byte
         STA    SUBADRL
         LDA    #>SUBROUTINE        ;High byte
         STA    SUBADRH
         PLA                        ;Restore ACC
         JSR    SUBSW
```

A JMPBS is identical except that the last instruction is JMP SUBSW. This routine vectors to the subroutine in such a way that when an RTS is encountered in the other bank, control automatically returns to the next instruction in this bank. As in ROMCALL's, status flags cannot be passed in this manner, but registers are undisturbed.

Using an assembler with Macro capability such as Big Mac, using this routine becomes a simple matter of entering:

>>> JSRBS.SUBROUTINE

Though these routines are not trivial, once their function is understood they can become the basis for creating much more powerful programs than could normally exist in only 2K of memory. Indeed, both GPLE and APU make extensive use of all of these techniques. Of course it is desirable to keep the amount of bank switching and Rom switching to a minimum, both from the standpoint of speed and code effeciency. Therefore, an approach which organizes your program in logical blocks is desirable.

To summarize, in order for a Rom to be compatible with the APU it should meet the following criteria.

1) The instruction at $C800 in the "low" bank should be a JMP to the routine handling Ampersand input. This routine should exit in the manner described under "Intercepting an Ampersand Command" above.

2) The instruction at $C803 in the "low" bank should be a JMP to the routine which prints a catalog of utilities available in the Rom. This routine should exit in the manner described under "Intercepting the & Catalog" above.

3) The byte at $CFFE must be $A5 in both banks of the Rom.

4) The Bankswitching subroutine described above should be at $CFE3 in both banks of the Rom. This is not necessary if no bank switching is required or 2716's are being used.

5) The protocol described above for bankswitching and executing JSRs between Roms should be followed for maximum effectiveness, although this is not necessary if no bank switching is required.

APPENDIX C - MEMORY MAP

Zero Page

$6 - 9	SAVED AND RESTORED AFTER USE
$F9 - FF	SAVED AND RESTORED AFTER USE
$26 - 27	PTR0 - used by bank switching subroutines
$2A - 2B	PTR1 - used by GPLE EDIT mode
$46	AAPU - temporary storage of A register (APU)
$47	XAPU - temporary storage of X register (APU)
$4A - 4C	CPTR - used by Ampersand catalog

APPENDIX C - MEMORY MAP

Page Three

$300	AMPTEST - $A5 if Ampersand vector has been set up
$301	MACTEST - $A5 if Macro table has been set up
$302	ROM STACK POINTER
$303 - 311	ROM STACK
$312 - 35A	GPLE VARIABLE SPACE & PARAMETER STORAGE
$35B - 378	RAM MACRO TABLE - DEFAULT
$379 - 37A	MACLO - pointer to beginning of Macro table - 1
$37B - 37C	MACHI - pointer to end of Macro table -2
$37D - 37F	ROMJMP - used for subroutine calls between Roms
$380 - 3B0	SUBSW - used for bank switching by subroutine calls
$3AC	SUBADRL - set up by calling program
$3AD	SUGADRH - set up by calling program
$3B1 - 3C6	GPLE VARIABLE SPACE & PARAMETER STORAGE

APPENDIX C - MEMORY MAP

$3C7 - 3C9	RESAVE - old RESET vector
$3CA - 3CB	AMPSAV - old Ampersand vector
$3CC	SLOT2 - $nO where n = slot of Rom board
$3CD	SLOT1 - $Cn where n = slot of Rom board
$3CE	NXTROM - used by ROMCALL to select next Rom
$3CF	ROMBNK - the currently active Rom

Driver Addresses

$CnOO	DRIVER - Initialize the Ultra Rom Board
$CnO3	RESIN - RESET vector points here
$Cn3D	GPINPUT - General purpose input vector
$Cn42	AMPVEC - Ampersand vector points here
$CnE4	CONNECT - Reconnect GPLE without initializing

I/O Space

$C800	ROMBADR - input to Ampersand Rom(s)

APPENDIX C - MEMORY MAP

$C803	CATBADR - input to Ampersand Catalog(s)
$CC98-CFE2	ROM MACRO TABLE (bank 0)
$CC88	SAVZ - Save free Zero Page addresses (bank 1)
$CC9F	RESZ - Restore free Zero Page addresses (bank 1)
$CFE3	BANKSWITCH - bank switching subroutine (all banks)
$CFFE	CHKBYT - must be $A5 (all banks)

APPENDIX D - SOURCE LISTINGS

```
                  ****************************************
                  *                                      *
                  *    DRIVER CODE FOR ULTRA ROM BOARD   *
                  *                                      *
                  *     (C) 1983 - HOLLYWOOD HARDWARE    *
                  *                                      *
                  *        WRITTEN BY:                   *
                  *                                      *
                  *                    PAUL JOHNSON      *
                  *                                      *
                  ****************************************

                               ORG  ROMBADR

C800: 38             DRIVER    SEC                  ;Main entry point
C801: B0 06                    BCS  DR1

C803: A9 03          RESIN     LDA  #>DOSWARM-1 ;Reset entry point
C805: 48                       PHA
C806: A9 CF                    LDA  #<DOSWARM-1
C808: 48                       PHA

C809: 20 58 FF       DR1       JSR  RETURN          ;Find out which slot
C80C: BA                       TSX                  ;We're in
C80D: BD 00 01                 LDA  STACK,X
C810: 8D CD 03                 STA  SLOT1           ;$Cn
C813: 0A                       ASL
C814: 0A                       ASL
C815: 0A                       ASL
C816: 0A                       ASL
C817: 09 80                    ORA  #%10000000
C819: 8D CC 03                 STA  SLOT2           ;$n0: n=slot+8
C81C: 85 26                    STA  PTR0            ;Used to turn rom on

C81E: A9 00                    LDA  #0
C820: 8D CE 03                 STA  NXTROM          ;Next rom to turn on
```

```
C823: 8D CF 03             STA  ROMBNK        ;Rom currently active

C826: A0 10                LDY  #$10
C828: 8C 02 03             STY  ROMSTK        ;Rom stack pointer

C82B: 99 B4 03   DRVLP     STA  HADRTBL-1,Y   ;Clear page 3
C82E: 88                   DEY                ;Address table
C82F: D0 FA                BNE  DRVLP

C831: A9 C0                LDA  #$C0
C833: 85 27                STA  PTR0+1
C835: AD FF CF             LDA  ROMOFF        ;Turn off all roms
C838: B1 26                LDA  (PTR0),Y      ;Turn on rom 0
C83A: 4C 7F C9             JMP  GPLEIN        ;Jump to init routines

C83D: F8         GPINPUT   SED                ;General purpose input
C83E: 18                   CLC                ;vector. Assumes that
C83F: 48                   PHA                ;NXTROM and ROMJMP have
C840: 90 14                BCC  COMMON        ;been set up by calling
                                              ;program
C842: 48         AMPVEC    PHA                ;Ampersand input vector
C843: A9 01                LDA  #AMPROML      ;$3F5.3F7 Point here
C845: 8D CE 03             STA  NXTROM
C848: F8                   SED
C849: 38                   SEC
C84A: B0 0A                BCS  COMMON

C84C: 18         CSWLGPLE  CLC                ;Character output vector
C84D: 24                   DFB  BITCODZ
C84E: 38         KSWLGPLE  SEC                ;Character input vector
C84F: D8                   CLD
C850: 48                   PHA
C851: A9 00                LDA  #0
C853: 8D CE 03             STA  NXTROM

C856: 98         COMMON    TYA                ;All routines come
```

```
C857: 48                 PHA                ;this way
C858: 08                 PHP
C859: AC 02 03           LDY  ROMSTK        ;Check rom stack
C85C: C0 11              CPY  #$11          ;for tampering.
C85E: B0 03              BGE  REHKGP0       ;Rehook if so
C860: 88                 DEY                ;Check also for overflow
C861: D0 10              BNE  STKOK         ;Branch if ok
C863: 68        REHKGP0  PLA                ;Rehook the rom card
C864: A9 00     REHKGP1  LDA  #0
C866: 85 25              STA  CV            ;Force screen clear
C868: 8D 01 03           STA  MACTEST       ;Reinit macros
C86B: 8D 00 03           STA  AMPTEST       ;Reinit ampersand
C86E: 68        REHKGP2  PLA
C86F: 68                 PLA
C870: 6C F2 03           JMP  (RESET)

C873: 8C 02 03  STKOK    STY  ROMSTK        ;Dec rom stack
C876: AD CF 03           LDA  ROMBNK        ;Get current rom
C879: 99 02 03           STA  ROMSTK,Y      ;Save on stack
C87C: AD CE 03           LDA  NXTROM        ;Get next rom
C87F: 8D CF 03           STA  ROMBNK        ;Make it current
C882: A8                 TAY                ;Index by next rom
C883: AD CC 03           LDA  SLOT2         ;$n0
C886: 85 26              STA  PTR0
C888: A9 C0              LDA  #$C0
C88A: 85 27              STA  PTR0+1
C88C: AD FF CF           LDA  ROMOFF        ;Turn off all roms
C88F: B1 26              LDA  (PTR0),Y      ;Turn on next rom
C891: 68                 PLA                ;Processer status

C892: AC FE CF           LDY  CHKBYT        ;Check to see if there
C895: C0 A5              CPY  #ROMBYT       ;is a rom in that slot
C897: D0 2C              BNE  DREXIT        ;If not, exit

C899: 48                 PHA                ;and vector to the
C89A: 28                 PLP                ;appropriate location
```

```
C89B: 29 08              AND  #%00001000
C89D: D0 11              BNE  DECMODE

C89F: 68                 PLA                 ;Restore registers
C8A0: A8                 TAY
C8A1: 68                 PLA
C8A2: B0 06              BCS  KSW1

C8A4: 20 FA C9           JSR  CSWLIN         ;Output
C8A7: 38                 SEC
C8A8: B0 15              BCS  COMRTN

C8AA: 20 1B CA  KSW1     JSR  KSWLIN         ;Input
C8AD: 38                 SEC
C8AE: B0 0F              BCS  COMRTN

C8B0: D8        DECMODE  CLD
C8B1: 68                 PLA                 ;Restore registers
C8B2: A8                 TAY
C8B3: 68                 PLA
C8B4: 90 06              BCC  GP1

C8B6: 20 00 C8           JSR  ROMBADR        ;Beginning of & rom
C8B9: 38                 SEC
C8BA: B0 03              BCS  COMRTN

C8BC: 20 7D 03  GP1      JSR  ROMJMP         ;General purpose

C8BF: 2C FF CF  COMRTN   BIT  ROMOFF         ;All routines come
C8C2: 48                 PHA                 ;back this way
C8C3: 98                 TYA
C8C4: 48                 PHA
C8C5: D8        DREXIT   CLD
C8C6: AC 02 03           LDY  ROMSTK         ;Restore rom stack
C8C9: B9 02 03           LDA  ROMSTK,Y       ;Turn last rom
C8CC: 8D CF 03           STA  ROMBNK         ;back on
```

```
C8CF: 30 0C               BMI  DREX2
C8D1: A8                  TAY
C8D2: AD CC 03            LDA  SLOT2
C8D5: 85 26               STA  PTR0
C8D7: A9 C0               LDA  #$C0
C8D9: 85 27               STA  PTR0+1
C8DB: B1 26               LDA  (PTR0),Y
C8DD: EE 02 03  DREX2     INC  ROMSTK
C8E0: 68                  PLA
C8E1: A8                  TAY
C8E2: 68                  PLA
C8E3: 60                  RTS               ;Return to caller

C8E4: 4E 1D 03  CONNECT   LSR  IOFLG        ;Intercept I/O hooks
C8E7: A0 4E               LDY  #<KSWLGPLE
C8E9: 84 38               STY  KSWL
C8EB: A0 4C               LDY  #<CSWLGPLE
C8ED: 84 36               STY  CSWL

C8EF: AC CD 03            LDY  SLOT1
C8F2: 84 39               STY  KSWH
C8F4: 84 37               STY  CSWH

C8F6: A0 00               LDY  #$00
C8F8: 8C 15 03            STY  CSWINDX      ;Set-up flags
C8FB: 8C 14 03            STY  KSWFLG
C8FE: 60                  RTS

C8FF: 4C 00 C8  ROMJMP1   JMP  ROMBADR      ;Image of subroutines
C902: 20 E3 CF  SUBSW1    JSR  BANKSWE      ;to be moved to $37D
C905: 08                  PHP
C906: 85 26               STA  PTR0
C908: 86 27               STX  PTR0+1
C90A: BA                  TSX
C90B: BD 02 01            LDA  STACK+2,X
C90E: C9 AD               CMP  #<SUBADRH
```

```
C910: D0 11     246             BNE  DOJSR
C912: BD 03 01  247             LDA  STACK+3,X
C915: C9 03     248             CMP  #>SUBADRH
C917: D0 0A     249             BNE  DOJSR
C919: 68        250             PLA
C91A: AA        251             TAX
C91B: 68        252             PLA
C91C: 68        253             PLA
C91D: 8A        254             TXA
C91E: 48        255             PHA
C91F: A9 4C     256             LDA  #JMPCODE
C921: D0 02     257             BNE  SUBOVER

C923: A9 20     259  DOJSR      LDA# JSRCODE
C925: 8D AB 03  260  SUBOVER    STA  SUBJSR
C928: A6 27     261             LDX  PTR0+1
C92A: A5 26     262             LDA  PTR0
C92C: 28        263             PLP
C92D: 20 FF CF  264  SUBJSR1    JSR  ROMOFF
                265  SUBADRL1   EQU  *-2
                266  SUBADRH1   EQU  *-1
C930: 4C E3 CF  267             JMP  BANKSWE

C933: AD 98 CC  269  ESCPNT1    LDA  ESCTBL       ;Used by macro
C936: 60        270             RTS
C937: 00        271  ENDRIV     DFB  0


                ****************************************
                *                                      *
                *   THESE ARE THE MACROS USED FOR BANK *
                *   SWITCHING AND INDIRECT JUMPS, ETC. *
                *                                      *
                ****************************************
```

APPENDIX D - SOURCE LISTINGS

```
        DO   0              ;Required by assembler

BANKSW  MAC                 ;This routine is at $CFE3
        PHA                 ;in both bank of all roms
        TYA                 ;It is used by JSRBS &
        PHA                 ;JMPBS to do the actual
        LDA  ROMBNK         ;bankswitching.
        EOR  #TOGBYT        ;Use #$80 to
        STA  ROMBNK         ;toggle high bit.
        TAY                 ;The fact that it's in
        LDA  #$C0           ;the same place in both
        STA  PTR0+1         ;banks means that
        LDA  SLOT2          ;when the actual
        STA  PTR0           ;bank switch takes place
        LDA  (PTR0),Y       ;<--HERE
        PLA                 ;the next instruction is
        TAY                 ;the same in both banks.
        PLA
        RTS
        DFB  ROMBYT         ;Test if rom is installed
        <<<

JMPBS   MAC                 ;JMP, BANK SWITCHING
        PHA                 ;Save A
        LDA  #<]1           ;]1 = The subroutine
        STA  SUBADRL        ;to be called
        LDA  #>]1
        STA  SUBADRH
        PLA
        JMP  SUBSW          ;Uses SUBSW @ $380
        <<<

JSRBS   MAC                 ;JSR, BANK SWITCHING
        PHA                 ;Variable ]1 is address
        LDA  #<]1           ;of subroutine.
        STA  SUBADRL
```

```
         LDA  #>]1          ;Also uses subsw @ $380
         STA  SUBADRH       ;so that an RTS at the
         PLA                ;end of subroutine will
         JSR  SUBSW         ;return to this bank.
         <<<

ROMCALL  MAC                ;Call subroutine in rom
         STA  PTR0          ;Save A
         STY  PTR0+1        ;Save Y
         LDY  #]1           ;Index to address table
         LDA  #]2           ;Rom number (0 - $F)
         JMP  ROMCON        ;Address of ROMCONT
         <<<

ROMCONT  MAC                ;Common to all ROMCALLS
         STA  NXTROM        ;Set up next rom
         LDA  ADRL-1,Y      ;Pick up address
         STA  ROMJMP+1      ;from address table
         LDA  ADRH-1,Y      ;and store in ROMJMP
         STA  ROMJMP+2      ;($37D    JMP  $XXXX)
         LDA  SLOT1         ;Push address of GPINPUT
         PHA                ;on to the stack
         LDA  #<GPINPUT-1   ;which will handle
         PHA                ;the actual rom switching
         LDY  PTR0+1        ;Restore Y
         LDA  PTR0          ;Restore A
         RTS                ;to GPINPUT
         <<<
         FIN
```

APB-102 QUICK REFERENCE GUIDE

GENERAL NOTES:

1) Escape functions are produced by hitting the "ESC" key and THEN the command key shown below. No RETURN is required, as the action takes place as soon as the command key is pressed.

2) Characters enclosed in square brackets are control characters (e.g. [A] = CTRL A). They are produced by holding the CTRL key down WHILE pressing the key in the brackets.

3) Commands marked with an asterisk (*) are repetitive, which means that after they execute they remain in the same mode. Examples are Escape functions like the cursor control keys "I,J,K,M" which can be used repetitively without having to hit ESC before each one.

4) The symbol "<cr>" is used to indicate a Carriage Return ([M]), either typed at the keyboard or embedded in the text.

APB-102 QUICK REFERENCE GUIDE

ESCAPE FUNCTIONS (ROM MACROS)

COMMAND KEY	FUNCTION
[B]	* B (prefix to LOAD, SAVE, RUN, as in BLOAD)
[C]	CATALOG
[D]	Prints GPLE version number
[E]	Duplicates [E] (Edit)
[F]	PEEK(A) + PEEK(A+1) * 256 (used by other MACROS)
[V]	VTAB PEEK (37) <cr> - moves up one line
[X]	Add a formatted REM statement to Basic program line (leaves you in insert mode.)
!	Print left bracket ([)
"	Print backslash (±)
#	Print underscore (_)
$	Print Ctrl left bracket ([[])
%	Print Ctrl backslash ([±])
&	Print Ctrl underscore ([_])
/	PRINT
*	Enter the monitor (CALL -151)
-	* L<cr> (list in MONITOR)
+	30 foreward spaces (used by RUN, SAVE, etc)
(	POKE -16368,0 (clear keyboard strobe)
)	PEEK (-16384) (check for key press)
0	Call -936 <cr> (HOME in Integer Basic)
1	Catalog,D1
2	Catalog,D2
4	Catalog,S4
5	Catalog,S5
6	Catalog,S6
7	CHR$(n) where you supply the n
8	CALL -868 (clear to end of line)
9	CALL -958 (clear to end of page)
L	LIST

APB-102 QUICK REFERENCE GUIDE

P	PRINT PEEK(n) where you supply the n
Q	Print value of the two byte number at the address pointed to by variable A
R	RUN <cr>
S	Print free sectors on last disk accessed (**ASSUMES A STANDARD DOS**)
T	TEXT:POKE -16300,0 (text page 1)
V	VTAB 1 <cr> (without clearing screen)
=	Display the Address and Length of the current RAM MACRO table
[SHIFT-M]	Save current RAM MACRO table to disk with the suffix = M$

[L]	LOAD <filename> on current screen line
[S]	SAVE <filename> on current screen line
[R]	RUN <filename> on current screen line
[Z]	DELETE <filename> on current screen line
X	LOCK <filename> on current screen line
Z	UNLOCK <filename> on current screen line

@		Clear screen and home cursor
A,B,C,D		Cursor moves, as per Apple manual
E,F		Clear keys, as per Apple manual
I,J,K,M	*	Cursor moves, as per Apple manual (Autostart Rom)
<	*	40 left arrows (affects data)
>	*	40 right arrows (affects data)

ARROW KEYS:

LEFT	*	Eight back spaces (affects data)
RIGHT	*	Eight foreward spaces (affects data)
UP	*	Same as I (Apple][e only)
DOWN	*	Same as M (Apple][e only)

APB-102 QUICK REFERENCE GUIDE

SYSTEM COMMANDS

NOTE:

All system commands except [W], [A], and RESET must be the first key typed after a <cr>.

COMMAND KEY	FUNCTION
[E]	Edit BASIC program line(s)
[R]	Reset Rom board, Warm Start. Easy way to turn off a printer or exit from monitor
[@]	Reset Rom board, Cold. Resets Macro table and rehooks Ampersand rom.
[W]	Edit line being typed. This can be used at any time if you are not already in the EDIT mode
[A]	Switch between upper and lower case entry. (If you start getting funny characters when you type, try hitting [A] twice.)
[C]	Returns to BASIC from the Monitor
[Q]	Quit GPLE and restore Apple to normal function
RESET	Reconnect GPLE (if active) and return to BASIC

APB-102 QUICK REFERENCE GUIDE

LIST CONTROL COMMANDS

COMMAND KEY	FUNCTION
[S]	Starts and stops any listing
SPACE	Single steps any listing
[P]	Lists next page (20 lines) of any listing
[C]	Cancels current listing and return to BASIC. (Works with CATALOG's also!)

THE EDIT MODE

NOTE:

Editing is invoked by typing [E] as the first key after a <cr>. The EDIT prompt will appear and you can enter the line numbers or parameters you wish to edit. The EDIT command can have the following forms:

1)	EDIT n1	Edits the line represented by n1
2)	EDIT n1,n2	Edits the range of lines from n1 to n2
3)	EDIT "ABC"	Edits all lines containing ABC
4)	EDIT "AB?"	Edits all lines containing a 3 character word starting with "AB"

5)	EDIT "ABC","DEF"	Edits all lines containing ABC and replaces with DEF. Stops at each line so that you may accept the change or restore the line.
6)	EDIT "ABC","DEF"/F	Fast search. Editor does not stop at each line.
7)	EDIT "ABC","DEF"/R	Raw search. Editor will find ABC even if it is embedded in a longer string.
8)	EDIT n1,,"AB?","DEF"	Any combination of parameters may be used so long as commas are included in the correct locations.
9)	EDIT .	Re-edits the most recently edited Basic line.
10)	EDIT ESC n	Edits or creates a Macro for the first key typed after the ESC key.
11)	EDIT ?	Lists the complete Macro table.

EDIT MODE COMMANDS

COMMAND KEY		FUNCTION
[I]	*	Inserts character(s) at cursor position
[D]		Deletes character(s) at cursor position

APB-102 QUICK REFERENCE GUIDE

[O]		Inserts next control character into text
[B]		Moves cursor to Beginning of line
[N]		Moves cursor to eNd of line
[F]	*	Finds the next character typed
[Z]	*	Zaps (deletes) all characters up to the next character typed
[R]		Restores Basic line to its condition before Editing began
[P]		Packs line, removing all spaces
[C]		Converts character at the cursor to the opposite case and advances the cursor
[Q]		Accepts the line up to the cursor and exits the Edit mode (This means you lose everything after the cursor)
[M]		Same as a carriage return or <cr>. Enters the entire line, regardless of the position of the cursor, and exits the Edit mode
[X]		Cancels any changes to the line and exits the Edit mode

AMPERSAND (&) FUNCTIONS

NOTE:

All Ampersand functions can be used in the Immediate mode (no program running). Several Ampersand functions can be used in Applesoft Basic programs, just like any other Basic command. An "ILLEGAL RUNTIME ERROR" will occur if you attempt to use an Immediate mode utility in a program.

COMMAND	FUNCTION
&	Catalogs the available ampersand functions. Immediate.
&[P]xxx	Passes the command "xxx" on to an Ampersand utility that was active when Rom Board was connected. Runtime.
&ASC	Returns the ASCII value of the next key pressed. Runtime.
&BIN	Prints the address and length of the last BLOADed or BRUN binary file. (**ASSUMES A STANDARD DOS**). Immediate.
&HIM	Prints the current value of HIMEM:. Immediate.
&LOM	Prints the current value of LOMEM:. Immediate.

&NOTNEW	Recovers from accidental use of NEW, FP, or INT. Program will have been damaged if a variable has been defined or a SYNTAX ERROR has occured. Immediate.
&SM"string",Rr,Ss	Searches all memory except $C000 to $CFFF (I/O space) for a match to "string." The high bit is ignored. If Rn is specified then bank "r" of the ram card in slot "s" is searched in the $D000 - $FFFF memory space. If no slot is specified then slot 0 is assumed. Immediate.
&SM"$hexadecimal",Rr,Ss	Same as above, except that it searchs for a list of hexadecimal numbers (e.g. &SM"$20 ED FD" would find all occurances of "JSR $FDED" in memory"). Spaces between numbers are not required if all numbers are two bytes long. Immediate.
&SPstring	Searches the program in memory for a match to string. Similar to the Raw search in GPLE. If you are looking for a word which may contain an Applesoft command (e.g. strONg) then the first character of the search string should be a quote. The quote will not be included in the search. Immediate.

&#	Returns the Hexadecimal and Binary equivalent of Decimal #. Immediate.
&$#	Returns the Decimal and Binary equivalent of Hexidecimal #. Immediate.
&%#	Returns the Decimal and Hexadecimal equivalent of Binary #. Immediate.
&MOVMAC	Moves the Macro Table to a location 256 bytes below the current HIMEM and relocates HIMEM to protect it. Immediate.
&RESMAC	Resets the Macro Table to Page 3 and moves HIMEM up 256 bytes. If the Macro table is already in Page 3 then no action will be taken. To be safe, this command should be used only if &MOVMAC has been used first. Immediate.
&DISCHR	Displays control characters as inverse characters. If GPLE is active, turn off with [R]. If not, use RESET. This is useful for finding control characters embedded in file names, programs, etc. Immediate.
&FRESEC	Prints the number of free sectors remaining on the most recently accessed disk. Assumes a standard DOS. Runtime.

&CEOL	Clears to end of line. Same as "CALL -868" Runtime.
&CEOP	Clears to end of page. Same as "CALL -958" Runtime.
&HGR	Same as HGR without clearing the screen Runtime.
&HGR1	Allows the selection of HGR page 1, but without text at the bottom. This is useful for switching back and forth between two full screens of graphics Runtime.
&HGR2	Same as HGR2 without clearing the screen Runtime.
&IF exp THEN exp : &ELSE exp	If the expression after the &IF is true then the expression(s) between THEN and &ELSE will be executed. Otherwise the expression(s) after the &ELSE will be executed. &IF / THEN / &ELSE structures can be nested but they must all be on one line. Runtime.
&PRINT USNG:string;variable(s)	Prints the Integer, Real, or Array variable(s) following the semicolon using the format of the string or string variable as a template. Runtime.

HOLLYWOOD HARDWARE, INC.

6842 Valjean Avenue
Van Nuys, California 91406

(213) 989-1204