DPX™

Development Pac Extension for the Sorcerer Computer

A CoResident Program that Extends the Command Capabilities of Exidy's DEVELOPMENT PAC.

By Don Ursem

(Requires Exidy's DEVELOPMENT PAC not included.)

A product of QS QUALITY SOFTWARE

Published by Quality Software.

INTRODUCING DPX

If you've tried programming using Exidy's new Z-80 DEVELOPMENT PAC system, you've found out that there are many excellent features packed into that 8K of ROM. But Exidy's design team opted to give you features like the Relocating Linking Loader and full I/O control, rather than a fancy editor. In fact, only 1K of the ROM PAC was left for the editor - and they put quite a few functions into that space.

What you didn't get were edit features like the ability to move the line pointer upward a few lines. Or to locate a particular word or symbol that needs changing. Or to make global string substitutions. DPX will allow you these editing capabilities and more.

There are also a number of frequently performed I/O operations that are not very convenient using the DEVELOPMENT PAC. It is not a trivial matter, for example, to exit the EDITOR, get into the MONITOR, set the baud rate for printing, then get back to DDT80 and set up for a printed assembly listing. With DPX, this kind of task is much easier.

DPX is a program that loads into memory right along with the DEVELOPMENT PAC. It adds in many convenience features that Exidy couldn't fit into the ROM PAC. DPX commands can be intermixed with the regular commands, all of which still work as usual.

The DPX commands were selected to take care of the functions that the author felt were most needed after working for many hours with Exidy's development system. With DPX loaded, you'll feel you have the power of a large scale computer development system at your command!

STARTUP PROCEDURE

1. Make sure your Sorcerer is TURNED OFF and insert your DEVELOPMENT PAC into the ROM PAC slot. Then power up the Sorcerer. This puts you in the DDT80 mode, indicated by the (.) prompt.

2. Enter the command:

```
E E003  <return>
```

This jumps you to the MONITOR so that you can load the DPX program from tape cassette. You should see the MONITOR prompt (>) on the display. There is a separate DPX version on the tape for your Sorcerer memory size. If you have 8K, 16K, or 32K, you can simply issue the MONITOR LOadGo command:

```
LOG DPXnn  <return>
```

where nn represents your memory size. That is, for a 16K machine, you'd load the file named DPX16 rather than DPX8 or DPX32. (If you have a 48K machine see the special instructions in Appendix A.)

On completion of the LOadGo, DPX takes control, prints its sign-on message, and places you back in DDT80 mode ready to go. That's all there is to it! All DEVELOPMENT PAC commands plus all DPX commands are now in operation.

3. At this point, use the M command to look at the I/O vectors. You'll see that DPX has already done its first chore for you. Since RAM based operation is the setup you'll probably want most often, DPX presets all I/O vectors for RAM based operation. If you require other settings, alter them at this time.

4. To get into the EDITOR mode after initial startup, it is best to use the command:

```
E :ED
```

This command initializes and clears the EDITOR workspace by resetting pointers for Start of Text (SOT), End of Text (EOT), and Low (LO) and High (HI) memory boundaries. It ensures that all boundaries are clearly known and that no garbage or left-over text is in your edit space. Subsequent re-entries to the EDITOR can be done using the DPX command #ED, or by the DDT80 command E :ER. These two commands do not reset text boundaries and do not destroy text already in the workspace. (The #ED command also performs some other 'housekeeping' tasks as you'll see later. So it is better to use #ED rather than E :ER).

COMMAND USAGE

DPX does not at all change the operational structure of the DEVELOPMENT PAC - it merely provides you with additional commands. It may be helpful at this point to review the structure of Exidy's development system. There are four operating modes:

1. DDT80 (Designer's Debugging Tool). This is the 'executive' mode of the DEVELOPMENT PAC. It is used to display and modify memory, to execute user programs, and to call the other modes of the DEVELOPMENT PAC. When in DDT80, a period (.) prompt is displayed on the video screen.

2. EDITOR. This is the mode that the programmer spends the most time in, creating and editing source files. (Some use it for simple letter-writing tasks too.) When in EDITOR, an asterisk (*) prompt is displayed.

3. ASSEMBLER. This mode is called only when the programmer is ready to assemble a source file into Z-80 object (machine language) code. The programmer is never really 'in' the ASSEMBLER, because control is always returned to DDT80 when the assembly is complete.

4. LOADER. This mode is called from DDT80 via the L command to load and link executable assembled modules. Only two commands (L and .) refer to this mode, and on exit control passes to the Sorcerer MONITOR. Like the EDITOR, the LOADER uses an asterisk (*) prompt.

All DPX commands, which are distinguished from DEVELOPMENT PAC commands in that they begin with a pound sign (#), may be entered from any mode. They can be used while in DDT80, when in the EDITOR, and even when in the Sorcerer MONITOR. But most DPX commands only make sense in and are intended for use from a specific mode.

The command summary on the last page of this manual groups both DEVELOPMENT PAC and DPX commands by operating mode. There is nothing to prevent the programmer from using DPX commands from other than the intended modes, but be careful. Several of the text-editing commands will bring unpredictable results if used from outside the EDITOR mode.

COMMAND SYNTAX

All DPX commands are distinguished from DEVELOPMENT PAC commands in that they begin with a pound sign (#) character.

The DPX commands are usually two letter mnemonics which if you prefer, may be spelled out as complete words. Thus #ED and #EDIT mean the same thing to DPX. Where parameters are required, these follow the command, separated by spaces. These conventions should be familiar to you, since Exidy MONITOR commands follow exactly the same rules.

As soon as you type a pound sign, the rest of that line, up to a carriage return, is interpreted as a DPX command. A second pound sign on that same line is treated as a restart of the DPX command, and anything previous to it is ignored.

Thus, if you make a mistake on the parameter value or start to enter the wrong command, just type another pound sign and keep going to enter the correct complete command.

Backspace is not supported within either DPX or DDT80 commands, because the commands are short and easily retyped. It only works with normal EDITOR commands or while entering text. If you do type a backspace (shift-RUB) it simply cancels the command, and DPX will print INVALID COMMAND.

DPX commands may be issued at any time, whether you are in DDT80 or EDITOR or the MONITOR. As soon as the pound sign is typed, the DEVELOPMENT PAC or MONITOR loses control until DPX is finished with its command. On completion, DPX feeds back just the carriage return, so that the system sees only what was typed <u>before</u> the pound sign (if anything) terminated by a return. If the DPX command is (as usual) the only entry on the line, the host system sees only a carriage return entered on that line.

In the MONITOR, this gets a reply of INVALID COMMAND. In the EDITOR mode, this simply prints the next line. In the DDT80 mode, nothing happens until you hit a second carriage return, and then you simply get another DDT80 prompt (.). DDT80 always expects two characters or more per command, so if you're in that mode and issue a DPX command, you'll have to hit two returns instead of one.

At first, you may forget to include the pound sign on DPX commands, which means they will be interpreted as invalid DEVELOPMENT PAC commands. This won't hurt anything. Just hit return and re-enter them properly. All DPX mnemonics were chosen so that they mean nothing to DEVELOPMENT PAC and will simply be ignored. (The only caution is that in the MONITOR, a #LOAD command without the pound sign becomes a LOAD command. You must not try to use this command outside of EDITOR mode anyway!)

<u>DPX TRANSFER COMMANDS</u>

If you have ever found yourself in the EDITOR, ready to exit, but with cassette I/O set when you really wanted RAM-based, or vice versa, you'll appreciate these commands. They allow you to directly jump from EDITOR to DDT80 or MONITOR level, perform any command, then jump back into EDITOR with no loss of text or change in I/O vector settings.

==

#MONITOR <return>   or #MON or #MO

==

This command does a direct jump to address E003H, which reenters the Sorcerer MONITOR. You can then load special printer or disk drivers, or do any MONITOR commands.

==

#DDT80 <return>   or #DDT   or #DD

==

This command jumps from EDITOR or MONITOR back into DDT80 mode. You can then reset I/O vector channels or do any DDT80 commands. This is different than the MONITOR PP command, which would reenter DDT80, but also would reset default RAM buffer boundaries. The #DDT80 command will not alter your currently set boundaries.

==

#EDITOR <return>   or #ED

==

This command jumps from MONITOR or DDT80 mode back into EDITOR mode. It also automatically resets the source input channel so that :SI=:BI. (This is so that after an assemble you do not inadvertantly reenter EDITOR with :SI still set to :BO.) If you are using cassette or any I/O other than RAM-based, you should return from DDT80 via E :ER or from MONITOR via the commands #DDT80 and then E :ER. For normal RAM-based development, you will find the #EDIT command very handy.

==

#ASSEMBLY <return>   or #AS

==

This command jumps to DDT80 exactly as does #DDT, but then sets up the :SI=:BO vector channel needed for RAM-based assembly. This permits the programmer to go immediately and directly from within EDITOR to prepare for an assembly. This command assumes RAM-based I/O so it will <u>not</u> wait to copy files to or from tape before it leaves EDITOR. Neither will it list out all of your text (a time saver if the text is long). This command does not trigger actual assembly. We suggest you issue #PH or #PR and then E :AS which will perform the actual assembly.

==
#RAMSET <return> or #RAM or #RA
==

This command does no jumps. It simply resets all I/O channels for RAM-based development. It may be used when in EDITOR or when in any other mode. (this command is issued automatically at DPX start-up time.)

DPX PRINTING CONTROL COMMANDS

DPX includes three commands to control printing. One is used to enable you to halt screen printing to examine text or assembly errors. The others permit use of a serial printer:

==
#PHALT <return> or #PH
==

This command enables you to halt any ASSEMBLER (or EDITOR) listing on the screen, by pressing any key while the listing is in progress. A second keypress of any key will resume listing. During a bad assembly, you may abort the assembly and exit to MONITOR by halting the listing and then hitting escape (ESC).

==
#PRINTER <return> or #PR
==

This command does a MONITOR SET T=1 and then routes all further screen output thru a built-in 300 baud driver for RS-232 serial printers. If you have a serial printer plugged into your Sorcerer's cassette/RS-232 connector, this is all you need to get assembly and EDITOR listings on hard copy.

If you are using 1200 baud serial printing or a parallel printer interface, you can customize this command in your copy of DPX. See CUSTOMIZING DPX INTERNALS (Appendix B).

==
#PVIDEO <return> or #PV
==

This command turns off your printer. It cancels the effect of a #PR or #PH and resets the baud rate to 1200 (SET T=0).

NOTE: You can use <u>either</u> #PV or #PR or #PH but not more than one at a time. These commands work through MONITOR internals, and are independent of the DEVELOPMENT PAC output channel vectors.

DPX EDITING COMMANDS

The following commands are intended to be used while in EDITOR mode. Most of them are locked out if you are inserting text and again become active upon return to normal EDITOR level. There is no way to prevent you using these in DDT80 or MONITOR, so they are active there as well. Those commands which affect pointer movement, however, may cause unpredictable results if you're not in EDITOR when you issue them.

```
=============================================================
         #UP nn <return>     or  #U nn
=============================================================
```

This command moves the EDITOR pointer upward. Now you can go back up a few lines without going all the way to beginning of file. The value of nn can be any number. This command moves the line pointer upward nn+1 lines, then the ending return causes EDITOR to drop back down one line and print it. If you omit the parameter (#U <return>) you'll move up one line, just as the normal EDITOR command of one carriage return moves down one line.

Because of the ending return, if you move upward to top of file, what gets printed is not the very first line, but the next one. This is like typing B<return>. The only way to print the very first line is to type BT<return>.

```
=============================================================
         #UP /string/     or #U /string
=============================================================
```

This is a companion command to the preceding one. It moves upward to the line upon which 'string' is found. This lets you find a previous line containing a specific word or value.

The search string may be any length up to a full line. It may not contain tab characters. The first character in 'string' must not be a space or any character less than ASCII zero (030H), or it will be ignored. (This is due to the use of internal monitor routines for parameter decoding.) Thus /-$/ won't be found but /F-$/ will. And #UP / ;comment/ is treated the same as #UP /;comment/.

If 'string' is not found, the pointer doesn't move up. Instead the next line is printed and you can search again. Search always begins above the current line so you can search and find the same string if it occurs in several lines.

```
==============================================================
          #AGAIN <return>        or #UA  or #A
==============================================================
```

Because repeated searches are very often needed to find the line you want, DPX always 'remembers' the latest used #UP /string/ command. By using the #A command, you will see a recap of that command, then it will be executed again.

Searching upward requires that you be positioned below the desired line. You can ensure that you are by entering B9999 <return> before beginning the search.

```
==============================================================
          #LINENO <return>       or #LI
==============================================================
```

All lines in the editor have an 'invisible' line number. When you enter a command such as:

```
B<return>6<return>
```

you are in effect moving the pointer to the 6th line, assuming that the very first line was numbered zero, the next (1), etc.

It is convenient to be able to tell what line you're at, and later to be able to return quickly to that line if you have moved away from it. The #LI command gives you this ability. It prints the 'number' of the current line but does not move the pointer. Later, you may return directly to that same line by typing B<return> then nn<return> where nn was the given line number.

(Notice that if you are at top of file, #LI will confirm you're at line 0, but then prints out line 1. Again, only the BT command can actually show you the very first line of text!)

===
#FIND /string/ <return>   or #F /string/
===

Since forward movement is so easily done by simply hitting one return then holding down the repeat key (thereby scrolling forward thru the text), you don't usually need any other forward search command. However in a long file, this takes several seconds, and since the #CHANGE commands always leave you at top of file, you'll sometimes want to return quickly down the file to a particular bit of text.

The #F command operates similar to the upward search command but searches forward. There are a few other differences as well.

#F will find strings with leading blanks or other special characters.

Also, with #F, the string may include wild characters. #F /LD ?,A/ would find either the line ' LD E,A' or the line ' LD D,A'. Wherever you use a question mark in the search string, any character in the text (including a question mark, of course) will qualify.

If the string is not found, #F takes you back to top of file. If it is found, you are left pointing at that line and can follow up with a #CHANGE request.

===
#CHANGE /string1/string2/ nn   or #C /str1/str2/
===

This is a GLOBAL CHANGE command. It will find all lines containing string1, print them, change string1 to string2, and reprint the changed line. It will continue to do so for 'nn' lines of text. It will then jump to top of file.

If no line count is given, only one change will be made, on the current line. It then jumps to top of file.

Only the first occurrence of string1 will be changed on each line. You can of course, reissue the command to change the next occurrence on each line.

#CHANGE expands or deletes text as required. A null string may be used as string2, (#C /string1//) in which case string1 will be deleted from the specified line(s). Tab characters may not be used - use blanks as needed. String1 may contain WILD characters (?) as in the #FIND command.

We suggest using #LI before each #C so that you can quickly return to the first changed line. Or you can use #F to get back there.

If you make a mistake while typing, hit @ or shift-RUB. Either will cancel the command.

The primary use of the #CHANGE command is not for line-by-line editing but to do things like change the name of a variable throughout a program or delete extra blanks or comments from a range of lines to free up more edit buffer space.

===

#QCHANGE /string1/string2/ nn   or #QC /str1/str2/

===

This performs exactly as the global #CHANGE command but is safer. It prints each found line then asks for a keyboard response telling it whether to make the alteration on that line. You may reply:

```
N   -- don't make this change, continue looking.
ESC --(escape key) don't change. Stop the search.
Y   --(or any other key) make the change and proceed.
```

DPX FILE HANDLING COMMANDS

The DEVELOPMENT PAC already includes the R and W file handling commands to permit moving (or 'spooling') large files of text through the EDITOR memory. These work well, but have limitations. The line-by-line I/O is time consuming for large files; indexing is not supported; you can only save from the beginning or merge onto the end of existing text; and you must exit the EDITOR to alter I/O vector settings.

DPX improves the DEVELOPMENT PAC file handling capability by adding:

1. fast file backup at 1200 baud,
2. the ability to extract or merge blocks of text at any point you specify, and
3. file names.

These features allow the programmer to build up a library of commonly used Z-80 subroutines that can easily be found and merged in when needed.

```
================================================================
          #STAT <return>        or #ST
================================================================
```

If you knew the low and high address boundaries of your text in memory, obviously you could quickly jump to MONITOR and SAVE the contents on tape, then jump back to EDITOR mode. Then you could later reload everything using the regular MONITOR LOAD command. (This is one way to back up and protect your current EDITOR session results against the system going 'down').

The #STAT command prints out four pointer addresses associated with the EDITOR buffer. These are start of text (LO), end of text (EOT), end of edit buffer (HI), and the address of the beginning of current line (LIN). It also tells how many bytes of text storage are still free.

Backup with DPX is easy. Just issue #STAT, then #MON. SAVE memory from LO to EOT on tape. Issue #ED to return to further editing. At any later point you can just issue #MON then LOAD back from tape and issue #ED to recover the entire previous EDITOR buffer contents. (You don't have to specify a reload address as long as the buffer boundaries are set the same way as when you did the SAVE). This method will save or reload 12k of text in under 2 minutes. (Baud rate is selected as for the #SAVE command, below. It is a good idea to always issue a #PV command before saving to be sure that the cassette baud rate is 1200, not 300 baud.)

#STAT can also help you fix up or recover lost text in case of a system glitch.

Here is how the EDITOR buffer pointers work. If you use the M command to examine these addresses, you'll see that normally the memory byte at the LO address contains zero (00H). Text actually begins at the next byte, with the end of each line denoted by ODH OAH (CR LF). The text runs continuously up to EOT, which contains another 00H value. From there up to HI is garbage or old text, invisible to the EDITOR commands. HI contains another 00H. HI is still 192 bytes below the real top of the EDITOR buffer, but it is the point at which the EDITOR will give you the FULL message.

If you insert a 00H value intentionally or through a sytem glitch between LO and EOT, the EDITOR will consider that to be EOT, and all text beyond it is 'lost'. Issuing #STAT tells you the exact current location of the EOT byte. If you change the value at that RAM location to a blank (020H), text will be 'recovered' up to the next (real or extraneous) 00H value.

If you've typed E :ED instead of E :ER, your text is not really gone. All you have to do is locate the new EOT, which should now be right next to LO, and change its value from 00H to 020H. Your text will reappear the next time you issue E :ER or #ED.

Finally, the LIN address shows you exactly where in RAM the currently displayed line begins, so you can find that text and alter it via the M command if you need to.

==

#SAVE fname nn x <return>   or #SA fname nn

==

This command saves part of the current text as a Named file designated by fname (up to 5 characters), writing the file to tape unit x (1 or 2). Saving begins at the beginning of the current line and proceeds for nn lines of text (nn is any decimal number up to 4 digits).

(You can see exactly what you are about to save by issuing nnT first, then saving nn lines).

All parameters are required, except for the tape unit number, which defaults to 1 if omitted.

Saving is done at the baud rate determined by the current MONITOR value of SET T=0 or SET T=1. If #PV is in effect, so is T=0, and baud rate is 1200. If #PR is in effect, so is T=1, and baud rate is 300. (Get in the habit of issuing #PV before using #SAVE).

Files are normal MONITOR format, with headers, and can be scanned with the MONITOR FILES command.

#SAVE is intended for extracting blocks of text which comprise useful subroutines, to build libraries of named routines on tape.

#SAVE can also be used for fast file backup of the complete edit buffer. But you must first ensure that you are at top of file, and must give a value of nn large enough to ensure that the entire text is saved.

On completion of a #SAVE, you remain in the EDITOR, positioned at the line following the first one saved.

If a mistake is made in specifying the file name, you can escape from a #SAVE by not hitting the (return) key to start the tape, but by hitting the (escape) key instead.

Do not use this command from MONITOR or DDT80 modes. It won't hurt your text, but the wrong area of RAM may be saved.

==
          #LOAD fname x <return>   or #LO fname
==

This command is not the same as a MONITOR LOAD command. A merge is performed when #LOAD is used. It opens the existing text at the current line, merges in a complete file from tape, then closes the existing text around it. This permits reload of routines stored on tape using the #SAVE command.

The new text is moved in just above the current line. With #LOAD you can quickly merge text at the top of file, end of file, or anywhere in the middle.

All parameters are required except tape unit number, which defaults to 1 if omitted.

#LOAD will read in any kind of MONITOR format file having the specified name. It will not read in unlabeled files created by the DEVELOPMENT PAC W command. It is up to you to be careful which files are which on your tapes.

Baud rate for #LOAD is affected by #PR, #PV, and direct monitor SET T= commands, just as for #SAVE. As usual, you cannot load files except at the baud rate used to save them. Issuing #PV before a #LOAD always ensures a baud rate of 1200.

If a mistake is made in specifying the file name, you can escape from a #LOAD by <u>not</u> hitting the (return) key to start the tape, but by hitting the (escape) key instead.

In case of tape CRC errors, #LOAD will exit to the MONITOR. You may retry the load using the normal MONITOR LOAD command, including start, end, and load address parameters. Just for this purpose, #LOAD always computes and prints the equivalent MONITOR command with these parameters, before it executes a load. All you have to do is copy that command in MONITOR to retry the load.

While #LOAD is being attempted, your text buffer is said to be 'open'. The part of the existing EDITOR text following the intended load point has been shifted higher in memory so that the end of the text is at HI and the intervening space has been filled with values of 02H. At the end of a successful #LOAD, the buffer 'closes' by shifting the remaining text back down in memory so that it immediately follows the file newly loaded. This 'splice' point is the first 02H value encountered in a quick scan from LO upward to HI.

Executing a #ED command will perform this 'close' procedure and get you back from the MONITOR into EDITOR level.

If you have not been able to successfully load the complete file, whatever part of it that did load will be merged into the text. You can delete these lines if you wish.

Finally, since #LOAD always loads a complete file, you have the responsibility of ensuring that there is sufficient space left in the EDITOR buffer to hold it along with the text already there. The #STAT command shows you free space in decimal, and you can compute it in Hex by using the H command to subtract EOT from HI.

When the loading begins, the required file space will be seen from the BLK parameter in the tape file header. If it won't fit, abort the load by pressing RUN STOP and then issue #ED to recover.

Normally, there is plenty of free space, and tapes load very reliably, so recovery procedure is rarely needed. It is explained just in case. (We have repeatedly loaded blocks of text as large as 22K with no difficulty).

Do not use this command except when in EDITOR mode.

DPX SPACE MANAGEMENT COMMAND

On a larger machine, it's easy to ensure lots of EDITOR buffer space simply by resetting RAM partition boundaries #3 and #4 downward. This leaves more room in the EDITOR partition, where you need it, and less in the object partition, where you usually only need several K of space. Using the DDT80 M command, you can set these down as far as they would default to on an 8K machine:

addr	value	description
0132H	7FH	top of DDT80 work area
0133H	0EH	
0134H	FFH	top of obj buffer; EDITOR LO
0135H	1CH	

DPX will not disturb these settings, and neither will any other command except the MONITOR PP command. (And with DPX transfer commands, you needn't use PP at all.) On a 32K Sorcerer, this lets you work in about 23K of EDITOR.

Nevertheless, you will one day devise a program so large, with so many routines, that you'll run out of RAM. In particular, the use of tabs, while it makes clean listings, eats up a lot of space for storing nothing but blanks. So, here is one last DPX command:

```
==================================================================
          #SQUEEZE <return>        or #SQ
==================================================================
```

This command runs through the text in a couple of seconds and removes all occurrences of multiple blanks. However, it always leaves 2 blanks between words, so the text is still easy to read and still assembles properly. It reclaims and frees up about 1/3 of the space previously taken up by the text!

You'll also find that this command speeds up the printing of listings quite a bit.

APPENDIX A:
48K OR OTHER CONFIGURATIONS OF DPX

Since DPX was developed on a 32K system, we could not directly provide load-and-go versions for a larger system. However, there is included on your tape a relocatable load module of DPX. If you have a 48K Sorcerer, or wish to use some other memory boundaries, reserving a portion of RAM for your own routines, the following one-time procedure will give you a load-and-go version tailored to your needs.

Set up DEVELOPMENT PAC with the following I/O channel:

```
:OI=:AO
```

Place the DPX cassette on tape unit 1, so that the load module copy will be read in. (The load module is on side 2 of the tape, the opposite side from where DPX8, DPX16, DPX32 are recorded.)

Enter the Sorcerer Monitor by typing

```
E E003 <return>
```

Load the load module by typing

```
LOG DPXLM
```

and pressing RETURN after starting the tape. The Load Module program will partition the development system for a 32K system. Create an object module for a 48K Sorcerer by typing

```
.L B452,BD40<return>
```

The value of BD40 is a dummy address for the Loader to use as its symbol table location. If you are using some other memory configuration, then instead of B452, calculate the load address for your system, where

```
addr=(top of partition 2)-8AE+1
```

The load command will read the tape from unit 1 and generate a loaded copy, giving you start and end addresses for a 48K machine, of B452H thru BCFFH. Type a period to escape to the Monitor. Now type the Monitor command SET X=B452 (or whatever your load address was). Now set up unit 1 to record and simply SAVE DPX48 B452 BCFF. This places a load-and-go version of DPX on tape. From now on, you need only use this tape copy, and LOG DPX48, just as for the other standard versions.

APPENDIX B: CUSTOMIZING DPX INTERNALS

If you wish, you may further tailor DPX to your particular needs. If you need special disk or printer drivers of your own, you may place the code for these in front of the DPX object code. At execution, DPX resets the top of partition 2 below itself, to protect against overwriting. The pointer for this is located at ORG+112AH, where ORG represents the start address of DPX. Change this value to the beginning of your routines and DPX will protect their RAM area also.

The command table is located within DPX at ORG+01DBH. It consists of a two-byte ASCII command followed by a two-byte jump address to the relevant routine, then another two-byte command, etc. You may change any jump address to point to your routine instead. Your routine should end with a RTN op code (0C9H).

If you are using a 1200 baud serial printer, you can modify the #PR command for 1200 instead of 300 baud operation. This is done by changing the value stored at ORG+040AH from 00H to 40H. If you are using a parallel printer, you will need to alter the value stored at ORG+0410H. Change the two-byte address stored there to the address of your own printer driver routine. If you have a Centronics-type printer, you can use the driver in the DEVELOPMENT PAC by setting:

ORG+0410H = 52H
ORG+0411H = C5H

After making such modifications, re-SAVE the customized version of DPX (plus your own code, if any).

For your information,

DPX version	ORG
DPX8	1452H
DPX16	3452H
DPX32	7452H

APPENDIX C: SUMMARY OF DPX AND DEVELOPMENT PAC COMMANDS

```
==================== EDITOR COMMANDS: ======================

       B                  - Move to Beginning (top of file)
       n<CR>              - Move line pointer down n lines
      #U n                - Move line pointer up n lines
      #U /str/            - Move up to string
      #F /str/            - Find string (downward search)
      #QC /str1/str2/ n   - Query/change n lines
      #C /str1/str2/ n    - Global change n lines
      #LI                 - Show current line number, text
       I                  - Insert lines above current
       nD                 - Delete n lines from current
       nT                 - Type n lines; don't move
       nR                 - Read n lines from :SI channel
       nW                 - Write n lines to :SO channel
       E                  - Exit, finish copying in or out
      #SA fn n x          - Save n lines onto x as file fn
      #LO fn x            - Load/merge file fn from x
      #ST                 - Statistics and free space
      #SQ                 - Squeeze out blanks
      #AS                 - Assembly :SI setup then #DD
      #MO                 - Goto MONITOR, no wait for copying
      #DD                 - Goto DDT80, no wait for copying

=============== MONITOR COMMANDS: ==========================

      PP                  - Reset partitions, reenter DDT80
      #DDT                - Goto DDT80
      #ED                 - Goto EDITOR, closing buffer


=================== DDT80 COMMANDS: ======================

       M                  - Memory display/modify
       R                  - Register display/modify
       E                  - Execute program (E E003  E :ED)
       H                  - Hex arithmetic calculator
       L                  - Loader
      #MO                 - goto MONITOR
      #ED                 - goto EDITOR (reentry)
      #RA                 - RAM based I/O channels
      #AS                 - Assembly I/O setup (:SI= :BO)
      #PH                 - Print with Halt option
      #PR                 - Print 300 baud RS-232
      #PV                 - Print 1200 baud video
```

APPENDIX D:
CAVEAT ON LOADING WITH DPX

DPX intercepts each keyboard input and writes into memory locations 0136H and 0137H the value for the ENDC = Top of Partition #2. This applies immediately after Loading and will corrupt a program loaded into that part of memory. To avoid this without hitting RESET, the user may enter the Sorcerer Monitor by typing

```
E E003 <return>
```

In the Monitor type

```
SET I=K <return>
```

This will disable DPX. To return to the DEVELOPMENT PAC with resetting partition boundaries type

```
GO C0E3 <return>
```

At this point DPX will not interfere with Loading into 0136-0137H. To later return to DPX, enter the Monitor and issue the command

```
GO ORG <return>
```

where ORG is the beginning address of DPX as defined in Appendix B.