print double sided and centered, staple, fold and then cut edges

Further Reading

For more information on the EBTKS module and its software navigate to the web site

http://www.fliptronics.com/EBTKS

https://tinyurl.com/HP8xForum

https://tinyurl.com/EBTKSVideo

For even more information see also

Dedicated to HP Series-80

http://www.series80.org

Information almost all HP Computer Systems and Accessories

http://www.hpmuseum.net/

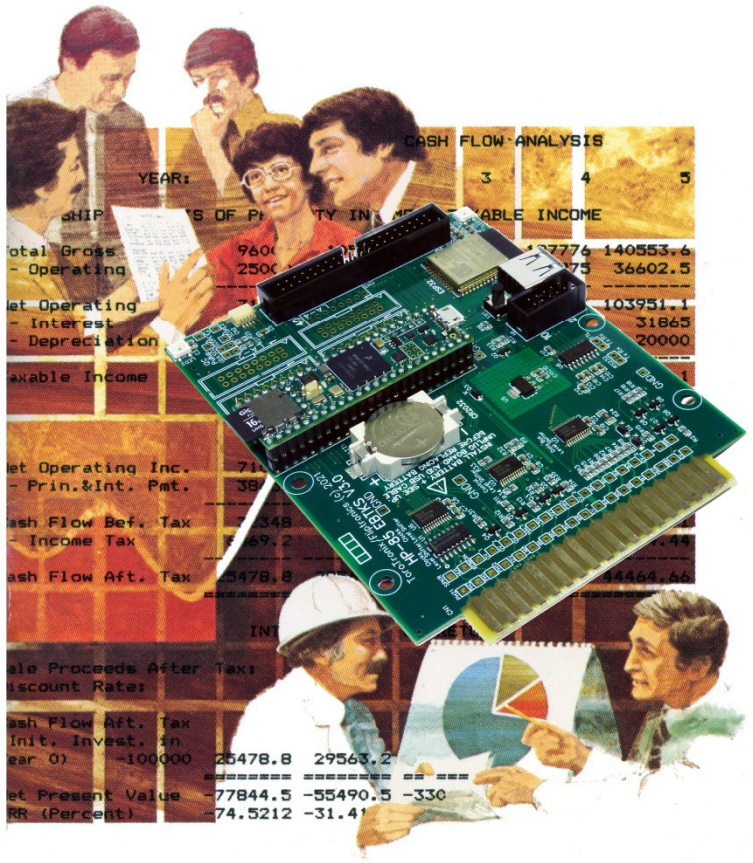
Discussion of HP Series-80 systems

https://groups.io/g/hpseries80

FLIPTRONICS / TOROTRONICS / KASER SOFTWARE

EBTKS

Plug-In Module Pocket Guide



Syntax Guidelines

DOT MATRIX	Items shown in Dot Matrix must be typed as shown; however, you can use lower case letters if you wish .
[]	Parameters enclosed in brackets are optional parameters.

Mass Storage Unit Specifier (MSUS)

The Series-80 systems use a so called MSUS to identify mass storage units like flexible disk drives, tapes or, hard disks.

Typical specifiers for usage with the EBKTS:

:D301	The disk drive unit 1, in a disk drive system with HP-IB address 0, attached to an interface having select code 3 (the EBTKS comes preconfigured to SC=3). This specifier is composed of SC*100 + HP-IB-Address*10 + Unit number.
:T	The tape drive. Can be a real working tape drive or one emulated by the EBTKS.
SDCard	The entire SDcard. Specific for the EBKTS.

Acknowledgements

The raw text of this Pocket Guide was originally written by Philip Freidin and Everett Kaser.

Martin Hepperle composed this booklet, streamlined the text and, created the title graphics, based on material which is Copyright © Hewlett Packard.

Created in 2021

\\	outputs a single ‚\‘ character
\r	outputs a CR character CHR$(13)
\n	outputs a LF character CHR$(10)
\t	outputs a TAB character CHR$(9)
\xHH	outputs a character specified by two HH hexadecimal digits
\nnn	outputs a character specified by three nnn octal digits

```
SPRINTF A$, "\t\r\n"
```

would achieve the exact same thing as

```
SPRINTF A$, "%c%c%c",9,13,10
```

You could also achieve the same thing by:

```
SDEOL 1
SPRINTF A$, "\t%s", SDEOL$
```

UNMOUNT msus$

The UNMOUNT statement prepares the given msus$ for removal.

```
UNMOUNT ":D300"
UNMOUNT ":T"
UNMOUNT "SDcard"
```

Use this variant before removing the SD card.

WSECTOR bufVar$, sector#, msus$

This statement writes a 256 byte sector to a drive.

bufVar$	buffer (dimensioned to ≥256 bytes).
sector#	sector index [0 to NumSectors-1].
msus$	the specification of the disk.

for the width specified.

[.precision] the maximum limit on the output, depending upon the 'type'.

Both [width] and [.precision] can either be a literal number included in format$ OR they can be the '*' character, in which case the '*' gets replaced by a number from the arg-list.

type a single letter indicating formatting of the item from the arg-list. type can be one of:

- i or d format the next numeric argument as a signed integer.
- u formats the next numeric argument as an unsigned integer.
- f or F formats the next numeric argument as a REAL in fixed-point notation. The difference is whether out-of-range numbers are output as upper or lowercase INF, INFINITY, or NAN.
- e or E formats the next numeric argument in standard "[-]d.ddd e[+/-]ddd" form. The difference is the case of the 'e' or 'E' used for the exponent.
- g or G format the next numeric argument in either fixed-point or standard-exponential format, whichever is more appropriate.
- x or X formats the next numeric argument as a hexadecimal value.
- o formats the next numeric argument as an octal value.
- s copies the next string argument into the output.
- c outputs a single character to the output. The argument may be the numeric value of the character, or the argument may be a string in which case the first character is output.

You can include special characters in the output by placing these character strings in the format$:

%% outputs a single ,%' character

Statements and Functions

AUXERRN

EBTKS issues custom error messages. These always have a system error number of 109. An ON ERROR handler routine can check the error number for 109, and if so, check ERROM for 241 (the decimal# of the AUXROM), and if so, it can then use AUXERRN to get a unique EBTKS value.

AUXREV

Return the AUX ROMs revision number.

BOOT

Re-boot the system, resetting both the Series80 computer and EBTKS. Any program that is in memory will be lost.

CRTROWS

Return the # of rows on CRT display (16 for HP85, 16 or 24 for HP87).

CRTCOLS

Return the # of columns on CRT display (32 for HP85, 80 for HP87).

CRTCURSOR state [,row, col]

Set the cursor to REPLACE (0) or INSERT (non-zero) mode, and optionally move the cursor to (row,col).

CRTCURSCOL

Return the current cursor column number.

CRTCURSROW

Return the absolute cursor row number.

CRTGETTOP

Return the current row# of display memory for the top row of the CRT display.

CRTON <0 | non-zero>

Blank or unblank CRT display (doesn't affect memory contents, just whether they're displayed or not).

CRTREAD$(row, column, len)

Read from CRT at (row, col) and returns a string of len characters from that location.

CRTSETTOP row

Set the CRT start row (0-63 on HP85, 0-53 (ALPHA) or 0-203 (ALPHALL) on HP87).

CRTWRITE string$, row, column

Write string$ to the CRT at (row, col). The value of row is absolute in CRT memory.

DATETIME Year, Month, Day, Seconds

Return four numeric values from the Real Time Clock.
Seconds are measured since midnight and are in [0 to 86399], the Day is in [1 to 31], the Month is in [1 to 12] and Year are in [1970 to 2038].

```
DATETIME A,B,C,D
DISP "Year";A;"Month";B;"Day";C
DISP "Hours";INT(D/3600);
DISP "Minutes";INT(FP(D/3600)*60);
DISP "Seconds";D-INT(D/3600)*3600-
INT(FP(D/3600)*60)*60
```

EBTKSREV$

Return a string containing the date and time of the EBTKS firmware build.

LED 2 is closest to the SD Card

The bits oft he first parameter select the LED:

1	LED 1
2	LED 2
3	both LEDs to the same color

Each of the three parameters R, G, and B is a number in the range 0 to 255. 0 is off, and 255 is full brightness for that color component.

SPRINTF dstVar$, format$ [, arg-list]

This keyword is similar to the C library sprintf() function. dstVar$ is the string where the formatted output will be stored and format$ is the format descriptor. The comma separated arg-list entries can be numeric or string expressions which match the corresponding elements specified in format$.
The SPRINTF keyword will output any normal text until it sees a '%' character. Then the next arg-list item is formatted according to the format descriptor:

```
%[flags][width][.precision]type
```

[flags]	optional, can be any of the following:
–	left-align output rather than right-align output (default)
+	prepends a plus for positive numeric types (the default does not prepend anything for positive values)
SPACE	(space character, not the word SPACE) prepends a space for the sign of positive values
0	if [width] is specified, prepends zeros for numeric types instead of spaces
#	select alternate forms for these types:
g, G:	leave trailing zeros
f, F, e, E, g, G:	always include decimal point
o, x, X	0, 0x, 0X respectively is prepended to non-zero numbers.
[width]	the minimum number of characters to output; no truncation of numbers too large

SDSLASH$

Return a character containing either '/' or '\'

SDSTORE fileName$

The SDSTORE statement stores the Basic program currently in memory to an SD file in tokenized form. A "LIF header" is included on the front of the file. This is essentially a shortcut way of doing

```
STORE "filename"
```

and then

```
SDEXPORTLIF "filename","SDname".
```

SDSTOREBIN fileName$

Store the Binary Program currently in memory to an SD file. A LIF header is included on the front of the image file.
This is essentially a shortcut way of doing

```
STOREBIN "filename"
```

and then

```
SDEXPORTLIF "filename","SDname".
```

SDWRITE src$, file#

The SDWRITE statement writes a string to an SD file. This is a binary mode file operation, no EOL-sequence is written. You can use SDEOL$ to format the output string with an EOL-sequence on the end.

```
DIM F$[128]
SDOPEN "ZINK",2,5
SPRINTF F$, "Hello, world!%s", SDEOL$
SDWRITE F$, 5
```

SETLED <1 | 2 | 3>, R#, G#, B#

Illuminate LEDs on the EBTKS.

LED 1 is closest to the Power Inlet

HELP [helpFile$]

The HELP statement displays help screen(s).

Some useful keys (see HELP NAVIGATION):

< and >	navigate to previous resp. next screen
↑ and ↓	select previous resp. next topic
END LINE	show selected topic
KEY LABEL, CONT	end HELP command

KBDBUFFER <0 | non-zero>

If the parameter is non-zero, takes over the keyboard and buffers up to 16 keys. If not, it turns OFF buffering and releases the keyboard.

KBDISKEY

The KBDISKEY statement returns 0 if the keyboard buffer is empty, otherwise it returns 1.

KBDKEY

The KBDKEY statement returns the next keycode from the buffer.

```
KBDBUFFE 1
IF KBDISKEY#0 THEN C = KBDKEY
KBDBUFFE 0
```

LISTROMS <0 | non-0>

If the supplied argument is 0, then a reference listing of all known Series 80 ROMs will be displayed on the CRT IS device. Otherwise only the ROMs that are currently present in the system will be listed.

MEDIA$(msus$)

This function returns the full path of the LIF image file associated with the given msus$

```
A$ = MEDIA$(":T")
A$ = MEDIA$(":D301")
```

MOUNT msus$, fileLIF$ [, modeFlag]

The MOUNT statement associates a specified LIF image file with the given msus$.

The optional modeFlag has the following effect:

0 if media exists, MOUNT, else error (default)
1 if media exists, MOUNT, else create blank and MOUNT
2 if media exists, error, else create blank and MOUNT

```
MOUNT ":D300", "NewDisk", 2
```

Errors if „NewDisk“ already exists, otherwise creates and MOUNTs it.

```
MOUNT ":T", "DataTape", 1
```

If „DataTape“ exists, MOUNT it, otherwise create a new file and mount it.

```
MOUNT "SDCard", "A"
```

Mount a new SD card (UNMOUNT the previous card first).

PEEK(address)

Return a byte from memory at address. The address is specified in decimal as is the returned value.

POKE address, byteVal

The POKE stamen writes a byte to the given address in memory. Both values specified in are decimal.

RPEEK(rom#, address)

Read a byte from the specified bank-switched ROM.

RPOKE rom#, address, byteVal

Same as POKE, except one of the AUX ROMs can be selected. These specific ROMs include 3 KB of RAM to communicate with the EBTKS module.

SDRDLINE dest$, bytesRead, maxBytes, file#

The SDREDLINE reads a line of text from the SD file into dest$ until one of the following occurs:
- maxBytes have been read (maximum: 1500),
- the end-of-file is reached,
- an EOL-sequence (CR/LF or LF or CR) is seen.

Sets bytesRead to the number of bytes read. The EOL-sequence is not stored in dest$. At end of file, bytesRead will be -1.

```
DIM A$[256]
SDOPEN "ZINK", 0, 5
SDRDLINE A$, L, 256, 5
```

SDSAVE fileName$

The SDSAVE statement saves the currently loaded program to an ASCII SD file. Each line ends with the current SDEOL sequence. Non-programmable.

SDSEEK(mode#, offset#, file#)

The SDSEEK function moves the “current position” pointer in an SD file to a specified location and returns the new file position.

The new location can be specified in one of three offsets as specified by the mode#:

0 absolute >= 0 offset from start of file
1 +/- relative offset from current point in file
2 absolute <= 0 offset from end of file

SDSIZE(fileName$)

Return the size of the specified SD file.

SDSLASH <0 | non-0>

Set the separator used in paths returned to the user.
SDSLASH 0 sets it to ‘/’ (the default)
SDSLASH 1 sets it to '\'.
This setting is remembered over a power cycle.

END of file. Used for appending to the file.

2 Truncate if it exists, otherwise create, position to START of file. Used for overwriting a file.

The given file# variable must be a number from 1 to 10. SD file access supports having up to 10 files open at the same time. These numbers are not related to the ASSIGN buffer numbers in BASIC.

SDPATH$(index#, path$)

Return one element of path$.
If index# is positive, 1 returns the 1st element, etc.
If index# is negative, -1 returns the last element, etc.
If index# is ONE greater beyond the number of elements, "" (zero-len string) is returned.
Otherwise, error.

SDREN oldName$, newName$

The SDREN statement renames the specified old file or directory to the new name.
It can also be used to move files between directories.

SDREAD dest$, bytesRead, maxBytes, file#

This statement reads from an open SD file, placing the bytes into dest$ and the number of bytes read into bytesRead. bytesRead will be either maxBytes or the remaining number of bytes in the file, whichever is smaller.The file# is the file number that was used to open the file with SDOPEN.

```
DIM A$[500]
SDOPEN "ZINK", 0, 5
SDREAD A$, L, 500, 5
```

SDRMDIR folderName$

The SDRMDIR statement removes a sub-directory from the SD drive. The sub-directory MUST be empty, or the SDRMDIR will fail.

RSECTOR bufVar$, sector#, msus$

This stamenent reads a 256-byte sector from a drive.

bufVar$	buffer (dimensioned to ≥256 bytes).
sector#	sector index [0 to NumSectors-1].
msus$	the specification of the disk.

SDATTR(fileName$)

The function SDATTR returns one byte with the attributes of the specified file in the lower two bits:
Bit 0: read-only, bit 1: subdirectory.

```
A = SDATTR("ZINK")
```

SDBATCH fileName$

Read characters from fileName$ and type them on the display. End-of- lines (CR/LF or LF) cause the END LINE key to be sent. Series 80 specific keys can be sent using the '\' character followed by the three octal digits of the keycode.
See the Assembler ROM manual (85 or 87, as appropriate) appendices for a complete list of keycodes.

SDCAT [fileSpec$]

The SDCAT statement displays a catalog of the SD file system.
fileSpec$ may include wild card characters (,*', ,?').
If no fileSpec$ is provided, ,*' is used.
As much of the file name as possible will be shown on the left of the screen and the filesize on the right. If the file name gets truncated, the last character will be underlined. If the file is READONLY, the size value will be underlined. If it is a sub-directory, a ,/' is shown for the size, and if it is a READ-ONLY sub-directory the ,/' will be underlined.

```
SDCAT
SDCAT fileSpec$
```

SDCD pathDir$

The SDCD statement changes the "current directory" to pathDir$, which may be a relative path or it may be absolute (starting with a '/').

SDCHAIN fileName$

Replace the current program with the program stored in fileName$ on the SD card and immediately execute it. The file must be created via SDSTORE. Otherwise the system will crash.

```
SDCHAIN "MODULE2"
```

SDCLOSE file#

The SDCLOSE statement closes an open file.

The file# paremeter is the file number from 1 to 10 that was used to open the file with SDOPEN.

```
SDCLOSE 5
```

SDCOPY source$, destination$ [, overwrite]

The SDCOPY statement copies the source$ file to the destination$ file. The overwrite flag can be:

0 overwriting an existing file is an error (default).
1 an existing file will be overwritten.

Does not support wild cards.
Does not support copying a complete directory.

SDCUR$

The SDCUR$ function returns a string containing the current absolute SD path.

SDDEL fileSpec$

The SDDEL statement deletes the specified regular file (no sub-directory). Wild cards ('*' and '?') may be used in the filename part of fileSpec$.

SDLOAD fileName$

The SDLOAD statement loads a Basic program into memory from an SD file that was SDSTORE'd.

SDLOADBIN fileName$

Load a Binary Program into memory from an SD file that was SDSTOREBIN'd or SDEXPORTLIF'd.

Caution
There is no file type checking. If you SDLOADBIN a file that was not SDSTOREBIN'd, or SDEXPORTLIF'd, the system may crash.

SDMKDIR folderName$

The SDMKDIR statement creates a sub-directory on the SD drive. If multiple sub-directories are included in "folderName$," all of the sub-directories except for the last one must already exist.

SDMORE source$ [, paginate]

Display the source$ file on the CRT, optionally "paginating" it, if paginate is 1. Output is NOT redirectable by the CRT IS statement.
When paging, the following keys are active:

RUN: display the rest of the file,
-LINE, BACKSPACE, -CHAR, PAUSE: terminate.

SDOPEN filePath$, mode#, file#

The SDOPEN statement opens a file on the SD card.

filePath$ may be an absolute path to the file or a path relative to the current folder (returned via SDCUR$ and set via SDCD).

mode# specifies how to open or create the file:

0 open for READ-ONLY, position to START of file, error if the file doesn't exist.
1 Open if it exists, otherwise create, position to

SDIMPORTLIF LIFname$, SDname$

The SDIMPORTLIF statement imports a file or a complete disk image from the SD card to a LIF disk.

Single file version

Import the file SDname$ from the SD Card. This file must be in LIF file format.
Such files may be created by SDEXPORTLIF (with headerFlag=1), but also by Everett Kaser's ASM85 program. The file is stored on a disk, which could be a physical disk, or a disk emulated by EBTKS.
LIFname$ can include an MSUS.
The LIF file format is prefaced with a 512 byte LIF header (a 256-byte Volume sector and a 256-byte Directory sector).

```
SDIMPORTLIF "MYPROG:D300", "/BPGM/p1.bin"
```

The file "p1.bin" stored in the directory "BPGM" on the SD Card will be copied to "MYPROG" on the disk "D300".

```
SDIMPORTLIF "data", "peoplesnames"
```

The LIF file "peoplesnames" will be copied from the current working directory on the SD Card to the file "data" on the current MASS STORAGE IS drive.

Whole Disk version

If the LIFname$ is just an MSUS, then the SDname$ file is expected to be a complete LIF disk image, such as SDEXPORTLIF would create if it is run with just an MSUS for the LIFname$.

```
SDIMPORTLIF ":D300", "disk2.lif"
```

The disk image "disk2.lif" will be copied from the SD Card to the MSUS ":D300".

SDIMPORTLIF is a DISK-ONLY statement, it will result in an error 115: INVALID MSUS if attempted on the ":T" tape drive.

SDEOF(file#)

The SDEOF function returns how many bytes from the current "file position pointer" to the "end of the file".

The return value is:

- 0 if the current file# position is at the end-of-the-file
- n the number of bytes from the current file position to the end of the file.

```
10 DIM A$[200]
20 SDOPEN "ZINK",0,5
30 SDRDLINE A$, L, 200, 5
40 DISP A$
50 IF SDEOF(5) THEN 30
60 SDCLOSE 5
70 END
```

SDEOL <0 | non-0>

The SDEOL statement selects the EOL sequence.

```
SDEOL 0      sets it to LF (the default)
SDEOL 1      sets it to CR/LF.
```

This setting is remembered on the SD card.

SDEOL$

Return a 1- or 2-character string with the current EOL sequence, either LF or CR/LF.

SDEXISTS(pathName$)

The SDEXISTS function returns a 1 if the specified file or sub-dir exists, otherwise it returns 0.

```
10 REM TEST FOR FILE
20 IF 0 = SDEXISTS("ZINK") THEN GOTO 100
30 ...
40 STOP
100 DISP "Files does not exist."
110 STOP
```

SDEXPORTLIF LIFname$, SDname$ [, headerFlag]

Export a LIF file from a LIF disk to a file on the SD card.

If the LIFname$ contains only an MSUS and no filename, then the entire disk is output, complete with volume and directory sectors and all files. The resulting image file can be used with the Series 80 emulator or with EBTKS on the SD card.

If headerFlag=0, then only the valid bytes from the LIF file are written. If headerFlag≠0, then a 512-byte LIF header (volume and directory sector) is written to the file, followed by all the sectors from the LIF disk that were allocated to the file.

SDEXPORTLIF is a DISK-ONLY statement, it will raise error 115: INVALID MSUS if attempted on the :T tape drive.

SDFFIRST fileName$, date$, size, attrib, fileSpec$

The SDFFIRST statement returns the first catalog entry that matches the fileSpec$. The statement SDFNEXT is used to return the following entries.

fileName$	The first name that matches fileSpec$ or a zero length string if none matches.
date$	The associated date and time
size	The file size (0 if a directory)
attrib	The file attributes. See SDATTR.
fileSpec$	The file name pattern to match. May include wild cards ("*" and "?").

SDFLUSH file#

The SDFLUSH statement forces any pending writes to be flushed to the SD card

The file# is a number from 1 to 10 that was used to open the file with SDOPEN. If the file# is 0, then all open files are flushed.

SDFNEXT fileName$, date$, size, attrib

Repeated calls of the SDFNEXT statement return successive catalog entries. When all have been returned, it will return a filename$ of length 0.
The SDFNEXT statement must set up by a call to SDFFIRST.
For the return values see SDFFIRST.

SDGET fileName$

The SDGET statement reads a file (usually a file that was created with an SDSAVE command), and parses the lines, attempting to recreate the original program from the ASCII listing of it.
Non-programmable.

NOTE 1:
SDGET does not do a SCRATCH, so the SDGET'd lines will be merged into the currently in-memory program.

NOTE 2:
Although SDGET will read in lines > 96 characters in length and parse them, keep in mind that when you list those lines on the screen, you won't be able to edit them on the screen. The maximum line length that SDGET will read in is 255 characters (which, of course, is WILDLY too long).

NOTE 3:
If you do an SDGET of assembly language code while in ASSEMBLER mode, you will see the lines echoed on the displayed, likely "inching their way" up the screen when lines with comments are encountered.

SDHOME$

The SDHOME$ function returns a string containing the original starting absolute SD path, the "home" directory. This is always '/' on EBTKS, but is a path on the Everett Kaser Series 80 Emulator.