EBTKS

Project Top

Table of Contents

Indices and tables

EBTKS Overview

The Origin story for EBTKS

(A first person perspective by Philip Freidin. July/August 2020)

This project started off as an attempt to repair the tape drive of an HP-85A desktop computer. Most of these tape drives (maybe all) have failed because the rubber capstan that moves the magnetic tape in the DC100/HP98200A cartridges have either disintegrated or turned into goo. My companion web site documents my Repair Adventures .

Eventually I decided that the repair of the tape drives and the tapes was a lost cause, so I decided that I would design a Solid State Tape Drive replacement that would be compatible with the existing HP85 operating system, and be responsive to all the same commands. I discussed this with my long time friend Russell Bull, and together we have co-designed the hardware and the firmware to make it all work. During our development work, we turned to Everett Kaser for guidance and he soon joined our team and is responsible for the AUXROMs, which added extensive functionality in the form of over 70 new keywords.

At 2:00 AM on 02/22/2020, I wrote the following to Russell via Skype (lightly edited for PC correctness.):

```
You know, if we put a Teensy 4.0 (600 MHz A7 Dual issue CPU !!!)
on an I/O board, it should be more than enough to bit-bang 1.6 us
bus cycles, not only could we do solid state replacement of the
tape drive, but some of the 2MB of FLASH could be 8 or more ROMS
(8KB each), and we could also use some of the 1 MB of SRAM to
implement the 16KB RAM module. Could also do ADC/DAC as discussed,
may as well do a RS232 serial interface, and expose I2C and SPI
and a proto area. USB comes along for the ride, Since it is
supported in Arduino land, could also offer it as a CPU offload
engine  :-) , using the 85 for keyboard, screen, and printer.  My
mind is a-buzz with possibilities, and most are just software that
can be done after the fact.  So BOM is for Teensy 4.0, PCB, level
shifters, RS232 cvt and connector, crappy ADC (only ~1 MSPS), no
DAC, but does have I2S and SPDIF....... CAN?, SD card, I2C and SPI
pins? parallel port pins? Some flashing lights :-), PWM? RTC.  I'm
done for this message..
```

Very quickly, Russell did the initial designed of version 1.0 hardware (with a different code name) that was the basis for our initial feature set discussions. We manufactured the PCB concurrently with an extender board design (for the HP-85 I/O bus) and had a working prototype by the end of March. We then spent about 3 months working on the firmware, and finding various issues due to our initial design decisions. Most of these were related to not having enough I/O pins on the Teensy 4.0 . Around late May we decided to redesign the board and remove multiple features that we realized we would be unlikely to take advantage of, but more importantly we changed the processor module to the just released Teensy 4.1 that you now see in EBTKS V2.0.

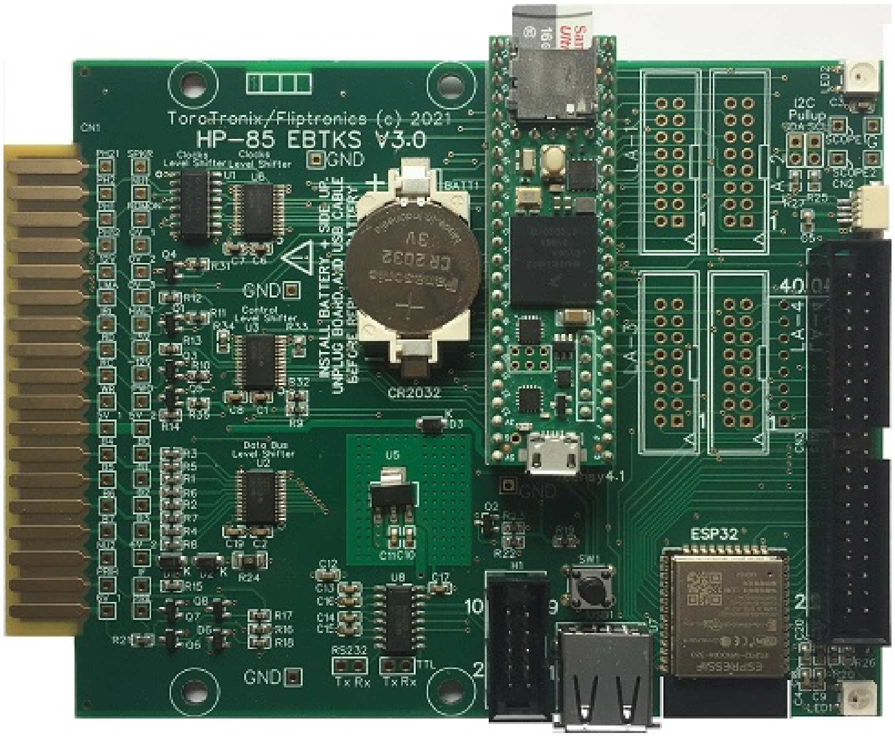
(Cont'd)

Along the way we had discussions with various people with more experience with the Series 80 computers (prior to this project neither Russell or I had ever used one), and by far the most helpful is Everett Kaser. He is now a valuable member of the team, and has worked on creating the AUXROM(s) that allow many of the features of EBTKS to be realized.

In August, Bob Armstrong joined in to do the first retro-computer add-on.

Like all pictures on this page, click the image for a larger version.

Production build of V3.0 EBTKS



Link to Prototype V2.0 used for most software development

EBTKS Feature List

Provided Services

- Tape drive emulation, using the 16 GB SD card to store Tape images. Theoretically the SD card could hold about 200,000 tape images. To use this service, the existing tape drive must be disabled by unplugging the two flat-flex cables.
- Disk drive emulation, using the 16 GB SD card to store Disk images. The number of disk images will depend on whether the emulated disk is a floppy or Winchester disk drive. We hope to support multiple disk drive types.
- RAM to fill the top 16 kB of the HP-85A.
- Up to 256 kB of Extended memory for HP85B (EDISK), and HP86/87 computers

(Cont'd)

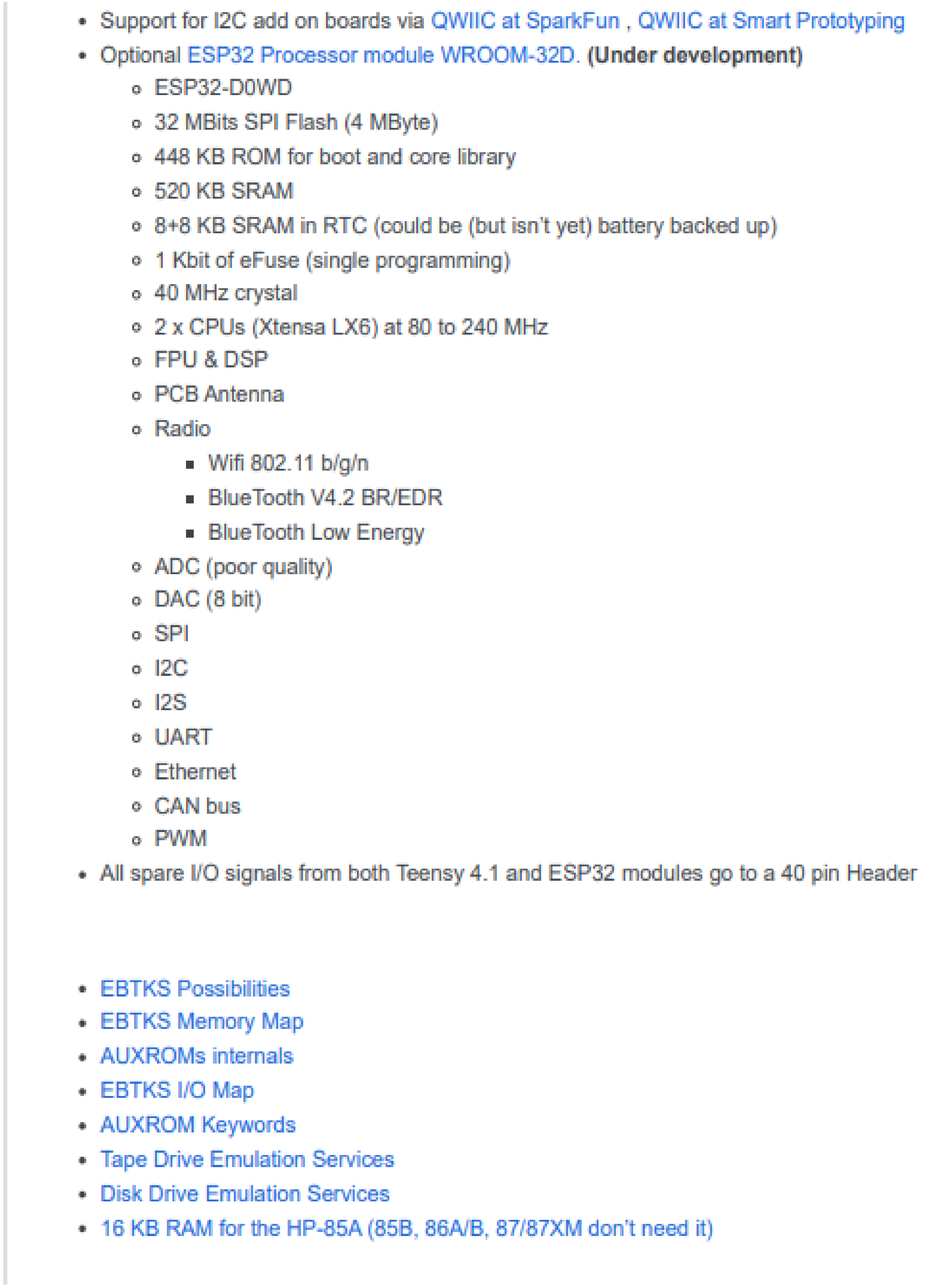
- EBTKS provides for up to 18 ROMs to be loaded from the SD card under control of the CONFIG.TXT file. Due to constraints imposed by the System ROMs, only 14 option ROMs are supported (one is system ROM 0). This is sufficient for all 12 standard published Option ROMs. The remainder of the 18 ROMs use a special facility for AUXROMs, discussed later. Currently there are four required AUXROMs which specifically supports EBTKS.
- No Jumper configuration. Configuration of options (like which ROMs are active, initial tape/disk image loaded) is under control of an easily editable CONFIG.TXT file on the SD card.
- Direct access to the CRT to move cursor and read/write text
- Improved auto-start program can include multiple commands or a simple batch file.
- Only requires 1 slot in the I/O backplane
- HELP facility (still in development)
- EBTKS Console using serial-over-USB
- Other features are still in development
- Over 70 New Keywords

AUXERRN	AUXREV	BOOT	CRTCOLS	CRTCURSCOL
CRTCURSOR	CRTCURSROW	CRTGETTOP	CRTON	CRTREAD$
CRTROWS	CRTSETTOP	CRTWRITE	DATETIME	EBTKSREV$
HELP	KBDBUFFER	KBDISKEY	KBDKEY	LISTROMS
MEDIA$	MOUNT	PEEK	POKE	RPEEK
RPOKE	RSECTOR	SDATTR	SDBATCH	SDCAT
SDCD	SDCHAIN	SDCLOSE	SDCOPY	SDCUR$
SDDEL	SDEOF	SDEOL	SDEOL$	SDEXISTS
SDEXPORTLIF	SDFFIRST	SDFLUSH	SDFNEXT	SDGET
SDHOME$	SDIMPORTLIF	SDLOAD	SDLOADBIN	SDMKDIR
SDMORE	SDOPEN	SDPATH$	SDRDLINE	SDREAD
SDREN	SDRMDIR	SDSAVE	SDSEEK	SDSIZE
SDSLASH	SDSLASH$	SDSTORE	SDSTOREBIN	SDWRITE
SETLED	SPRINTF	UNMOUNT	WSECTOR	AUXBUF$
AUXCMD	AUXOPT$			

Hardware Resources

- 600 MHz Cortex M7 processor
  - Processor is an NXP IMXRT1062
  - Processor module is a Teensy 4.1
  - 1 MB high speed On-Chip RAM
  - 8 MB medium speed RAM external to the processor
  - Device USB port used for firmware update and diagnostic serial
  - Host USB port for USB Keyboard and maybe other devices
- 16 GB SD Card
- Two multi color LEDs

(Cont'd)

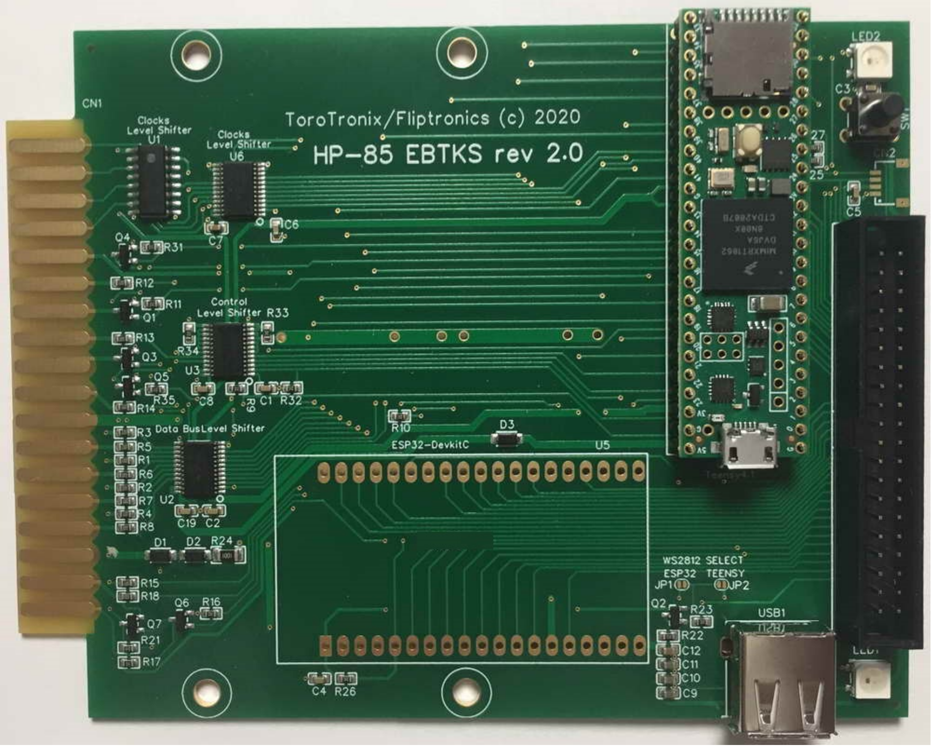


- Support for I2C add on boards via QWIIC at SparkFun , QWIIC at Smart Prototyping
- Optional ESP32 Processor module WROOM-32D. **(Under development)**
    - ESP32-D0WD
    - 32 MBits SPI Flash (4 MByte)
    - 448 KB ROM for boot and core library
    - 520 KB SRAM
    - 8+8 KB SRAM in RTC (could be (but isn't yet) battery backed up)
    - 1 Kbit of eFuse (single programming)
    - 40 MHz crystal
    - 2 x CPUs (Xtensa LX6) at 80 to 240 MHz
    - FPU & DSP
    - PCB Antenna
    - Radio
        - Wifi 802.11 b/g/n
        - BlueTooth V4.2 BR/EDR
        - BlueTooth Low Energy
    - ADC (poor quality)
    - DAC (8 bit)
    - SPI
    - I2C
    - I2S
    - UART
    - Ethernet
    - CAN bus
    - PWM
- All spare I/O signals from both Teensy 4.1 and ESP32 modules go to a 40 pin Header

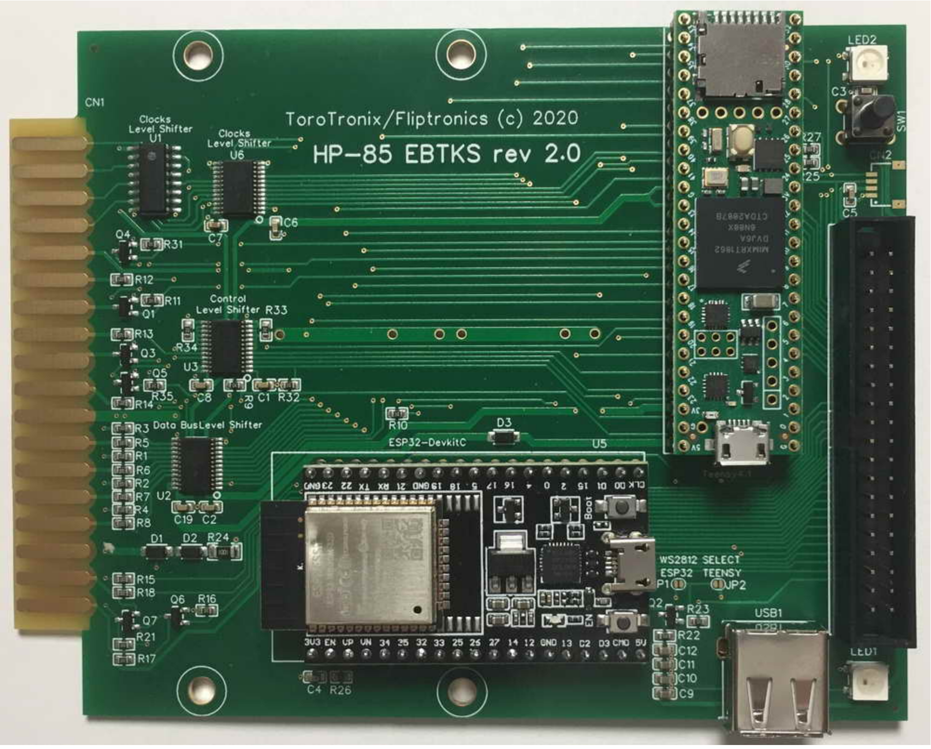
- EBTKS Possibilities
- EBTKS Memory Map
- AUXROMs internals
- EBTKS I/O Map
- AUXROM Keywords
- Tape Drive Emulation Services
- Disk Drive Emulation Services
- 16 KB RAM for the HP-85A (85B, 86A/B, 87/87XM don't need it)

Prototype V2.0 used for most software development

Prototype V2.0 EBTKS standard configuration
(image is missing the SD Card and the QWIIC connector)



Prototype V2.0 EBTKS with optional ESP32 module
(image is missing the SD Card and the QWIIC connector)



EBTKS Getting Started

Guide Rails

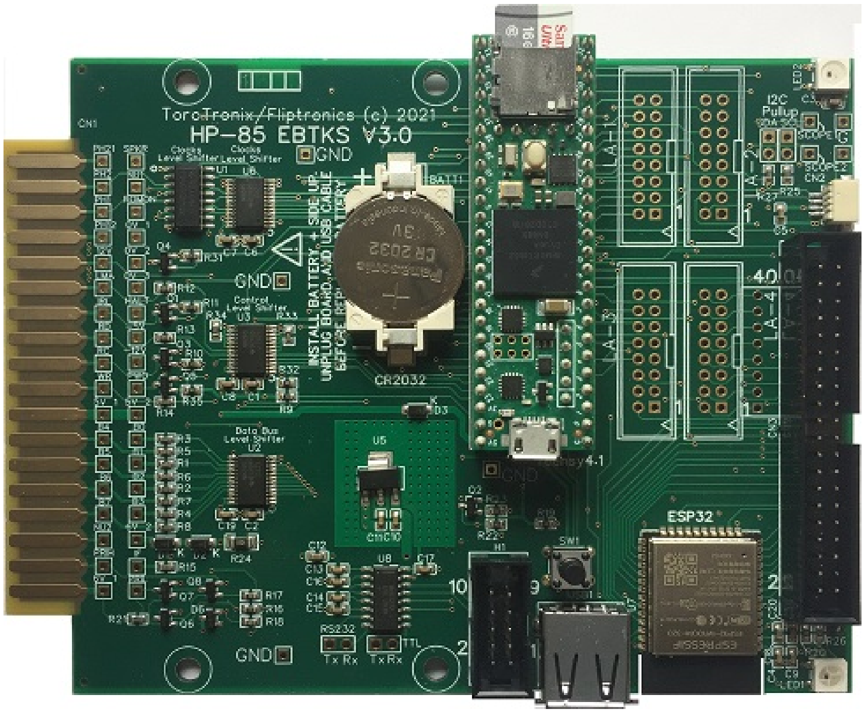
PVC or HDPE plastic guide rails are CNC machined out of gray PVC type 1 or black HDPE. The left and right guide rails are sized to match the guide slots in the backplane of your Series80 computer. These come standard with your EBTKS, unless you specify that you don't need them because you intend to use a 3D printed case.

3D Printed Case¶

The design files for a 3D printed case are now available on this page. Getting the case printed will be your responsibility, the EBTKS team does not provide these cases

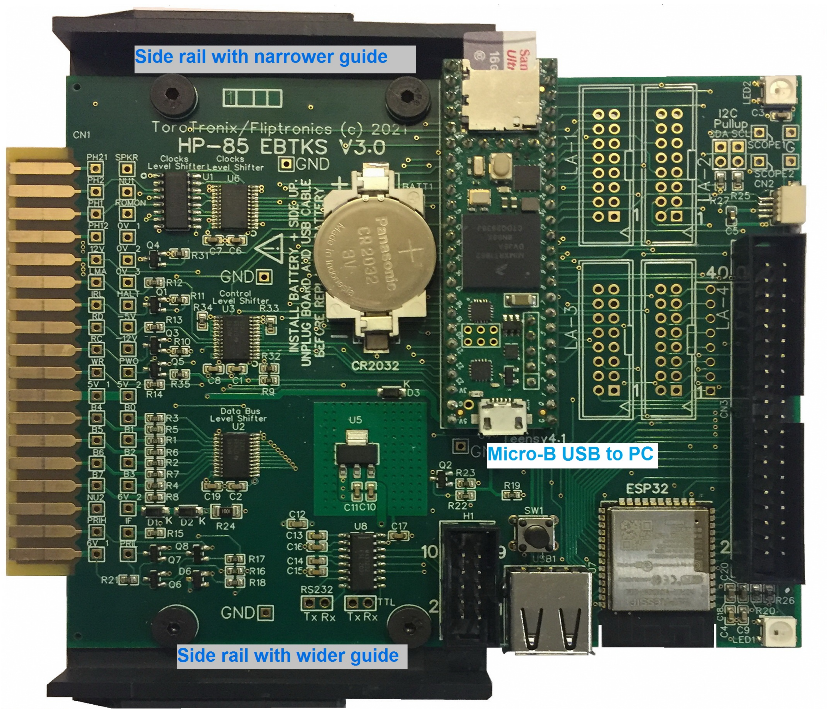
What's in the box

Depending on how you ordered your EBTKS, it is supplied as just the assembled Printed Circuit Board (PCB) or the PCB with Guide Rails attached



EBTKS without Guide Rails

(Cont'd)


Side rail with narrower guide
ToroTronix/Fliptronics (c) 2021
HP-85 EBTKS V3.0
Micro-B USB to PC
ESP32
Side rail with wider guide

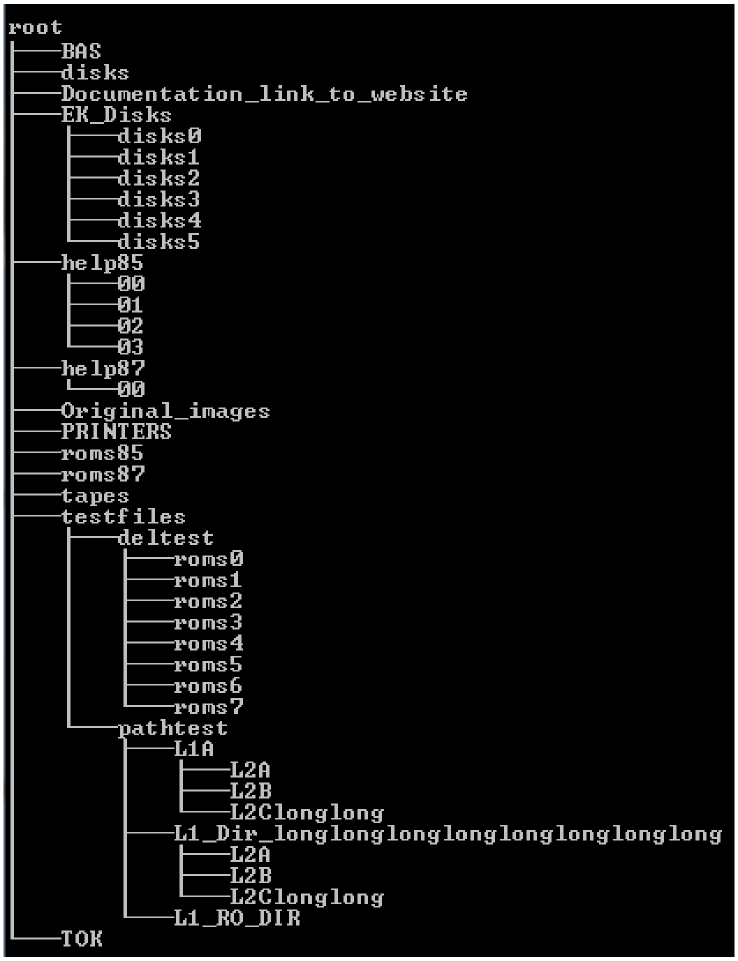
EBTKS with Guide Rails

There will be a 16 GB MicroSD Card preloaded with software installed in the Teensy 4.1 module. In case you may want to do additional development on the Teensy module, or maybe remove it from EBTKS to use in another project, the standard Teensy 4.1 reference card is included. If you do unplug the Teensy module (never needed for normal EBTKS use), do it very carefully by slowly levering up each end of the module with a plastic tool, to avoid damaging the module and the PCB beneath it. The pins on the Teensy 4.1 module are very thin and easily bent or broken.

EBTKS is Pre Configured

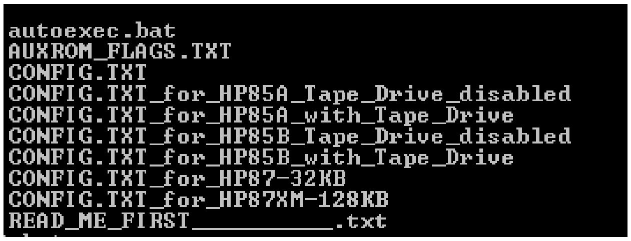
Based on information you provided at the time of your order, EBTKS has been preconfigured to match your system. This is done with the CONFIG.TXT file which you can edit to change configurations. Please note that this file has a very specific format, and if you edit it, you must verify that you do not mess up the formatting. On the just referenced page, there is information about a syntax checker that is easy to use, to check your changes. There are no user jumpers on EBTKS, every configuration option is controlled by CONFIG.TXT, which is located in the root directory of the MicroSD Card.

(Cont'd)



```
root
├───BAS
├───disks
├───Documentation_link_to_website
├───EK_Disks
│   ├───disks0
│   ├───disks1
│   ├───disks2
│   ├───disks3
│   ├───disks4
│   └───disks5
├───help85
│   ├───00
│   ├───01
│   ├───02
│   └───03
├───help87
│   └───00
├───Original_images
├───PRINTERS
├───roms85
├───roms87
├───tapes
├───testfiles
│   ├───deltest
│   │   ├───roms0
│   │   ├───roms1
│   │   ├───roms2
│   │   ├───roms3
│   │   ├───roms4
│   │   ├───roms5
│   │   ├───roms6
│   │   └───roms7
│   └───pathtest
│       ├───L1A
│       │   ├───L2A
│       │   ├───L2B
│       │   └───L2Clonglong
│       ├───L1_Dir_longlonglonglonglonglonglonglong
│       │   ├───L2A
│       │   ├───L2B
│       │   └───L2Clonglong
│       └───L1_RO_DIR
└───TOK
```

MicroSD Card Directory Tree



```
autoexec.bat
AUXROM_FLAGS.TXT
CONFIG.TXT
CONFIG.TXT_for_HP85A_Tape_Drive_disabled
CONFIG.TXT_for_HP85A_with_Tape_Drive
CONFIG.TXT_for_HP85B_Tape_Drive_disabled
CONFIG.TXT_for_HP85B_with_Tape_Drive
CONFIG.TXT_for_HP87-32KB
CONFIG.TXT_for_HP87XM-128KB
READ_ME_FIRST__________.txt
```

MicroSD Card Root Directory

Moving EBTKS to a different computer

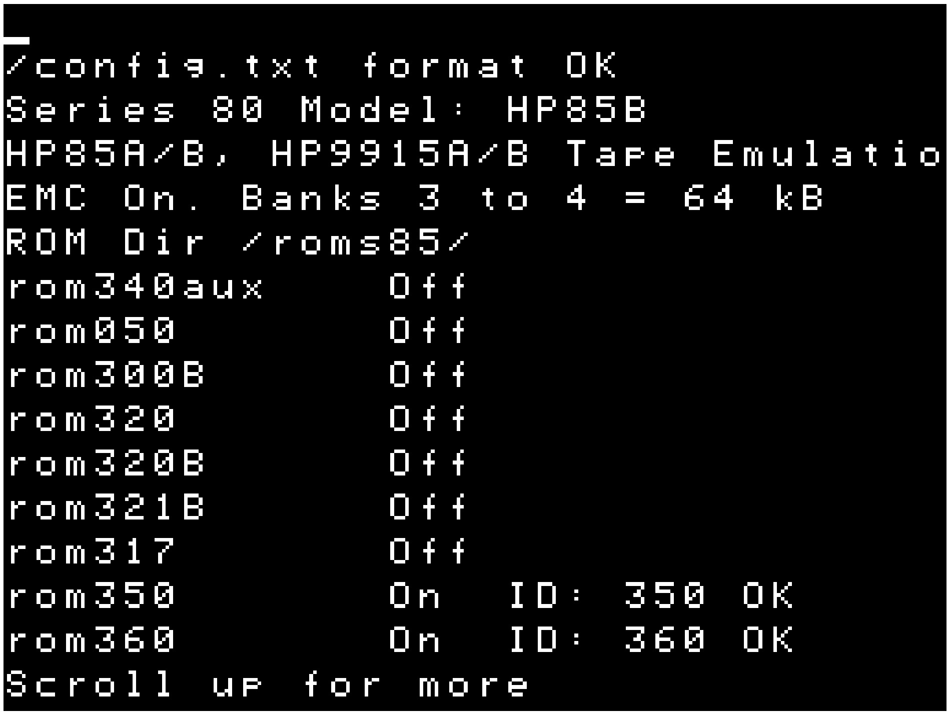
In the root directory, as well as the CONFIG.TXT file, there are 6 other configuration files, with hopefully informative names. If you want to change the computer your EBTKS is plugged into another computer that is not the same model and same configuration, the CONFIG.TXT file will need to be changed to match the new computer. The easiest way to do this is to delete the CONFIG.TXT file, and make a copy of one of the other 6 files, naming the copy CONFIG.TXT. You should never edit these 6 files, only copies of them. If you do mess things up, you can always get a pristine copy of the MicroSD Card image from the Download page

Time to start playing

(All of the following assumes the standard MicroSD Card file set. The example screen displays are from a HP85B.)

EBTKS is preconfigured to provide 5 floppy disk drives and one 5 MB Winchester drive (all emulated, the actual storage is files on the MicroSD Card).

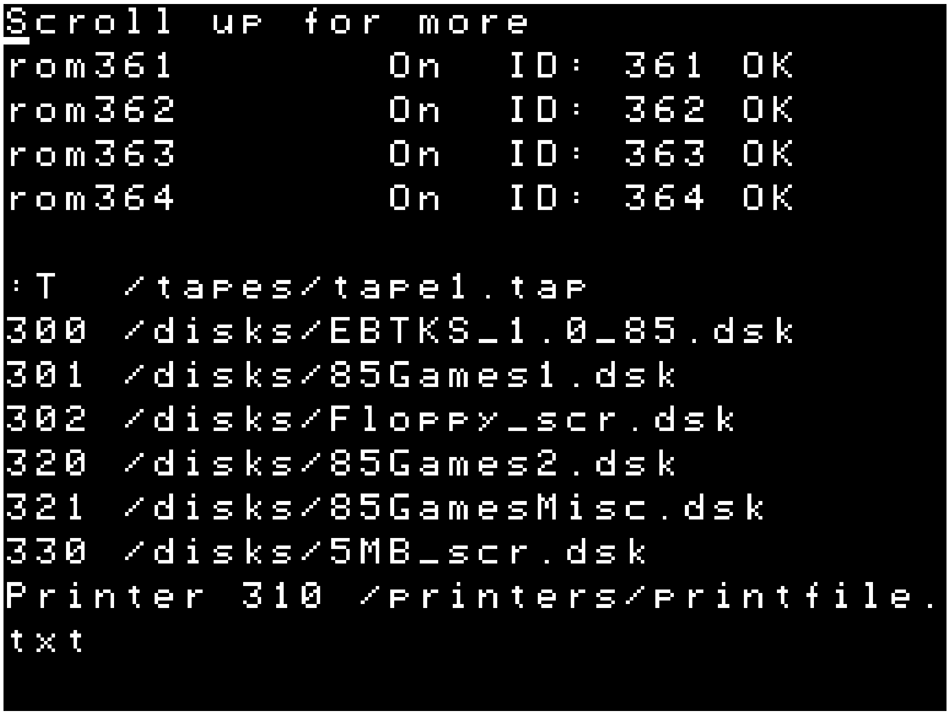
When you start your computer with EBTKS for the first time, it will show the following two screens (the second one by using the Shift-Roll to scroll up)



```
/config.txt format OK
Series 80 Model: HP85B
HP85A/B, HP9915A/B Tape Emulatio
EMC On. Banks 3 to 4 = 64 kB
ROM Dir /roms85/
rom340aux     Off
rom050        Off
rom300B       Off
rom320        Off
rom320B       Off
rom321B       Off
rom317        Off
rom350        On   ID: 350 OK
rom360        On   ID: 360 OK
Scroll up for more
```

First page

(Cont'd)



```
Scroll up for more
rom361       On  ID: 361 OK
rom362       On  ID: 362 OK
rom363       On  ID: 363 OK
rom364       On  ID: 364 OK

:T  /tapes/tape1.tap
300 /disks/EBTKS_1.0_85.dsk
301 /disks/85Games1.dsk
302 /disks/Floppy_scr.dsk
320 /disks/85Games2.dsk
321 /disks/85GamesMisc.dsk
330 /disks/5MB_scr.dsk
Printer 310 /printers/printfile.
txt
```

Second page

These two pages show how the CONFIG.TXT file was processed as the system started, and you should review the display carefully if you make any changes to CONFIG.TXT . Briefly it shows the following

- Starting on the first page above, for which model of Series80 computer the CONFIG.TXT file is configured.
- Whether emulation of the tape drive is enabled (the original goal of EBTKS). Note that for tape drive emulation to work correctly, the physical tape drive must be disabled by unplugging the two flat flex cables.
- Whether EBTKS provides extended memory, and how much
- Whether ROMs are loaded for HP85 (or 83 or 9915) or for HP86/87
- Which ROMs are enabled
- Seen on the second page above, the association between Mass Storage Unit Specifiers (abbreviated in HP Mass Storage Manual as msus) and the files that store the Floppy or Winchester disk image.

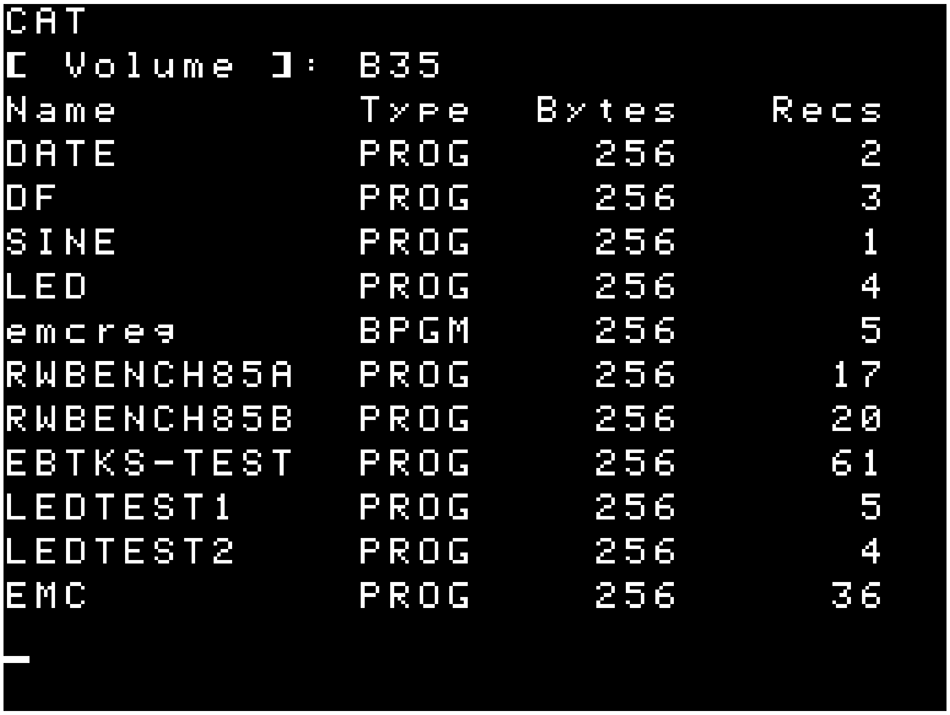
This startup display can be disabled in the CONFIG.TXT file by changing the line:

```
"CRTVerbose": true,
to
"CRTVerbose": false,
```

The disk image file associated with msus 300 is the initial default Mass Storage device (in the EBTKS situation, an emulated device), which is shown on the second page above is **/disks/EBTKS_1.0_85.dsk**

Typing **CAT** will list the contents of this emulated floppy disk (/disks/EBTKS_1.0_85.dsk)

(Cont'd)



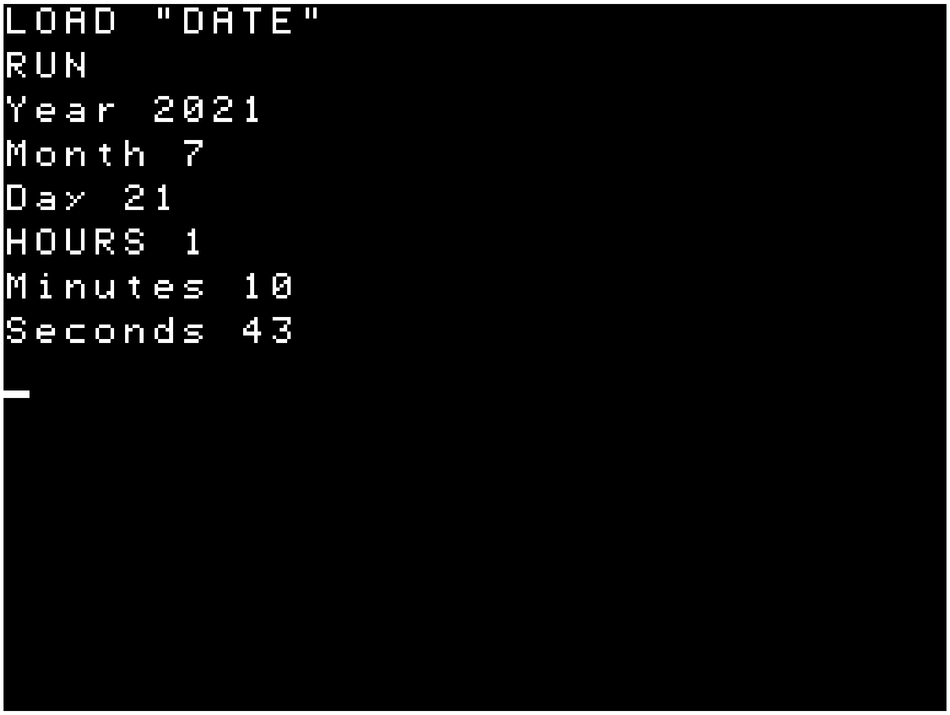
```
CAT
[ Volume ]: B35
Name         Type   Bytes    Recs
DATE         PROG     256       2
DF           PROG     256       3
SINE         PROG     256       1
LED          PROG     256       4
emcreg       BPGM     256       5
RWBENCH85A   PROG     256      17
RWBENCH85B   PROG     256      20
EBTKS-TEST   PROG     256      61
LEDTEST1     PROG     256       5
LEDTEST2     PROG     256       4
EMC          PROG     256      36
```

Catalog of msus 300

Some of these programs will run without problems, and some will need editing to match your computer's configuration

Here are some simple examples:

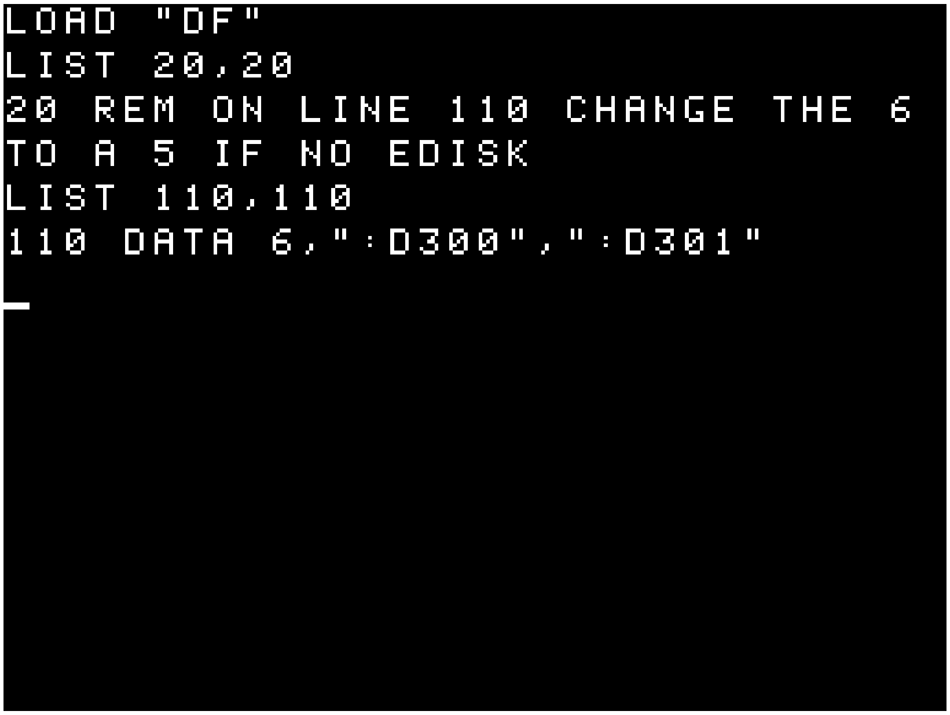
The DATE program retrieves the current time and date from the on-board battery backed up clock. It is set to California time, PDT time zone. It uses a new keyword DATETIME . LIST the program to see how it works.



```
LOAD "DATE"
RUN
Year 2021
Month 7
Day 21
HOURS 1
Minutes 10
Seconds 43
```

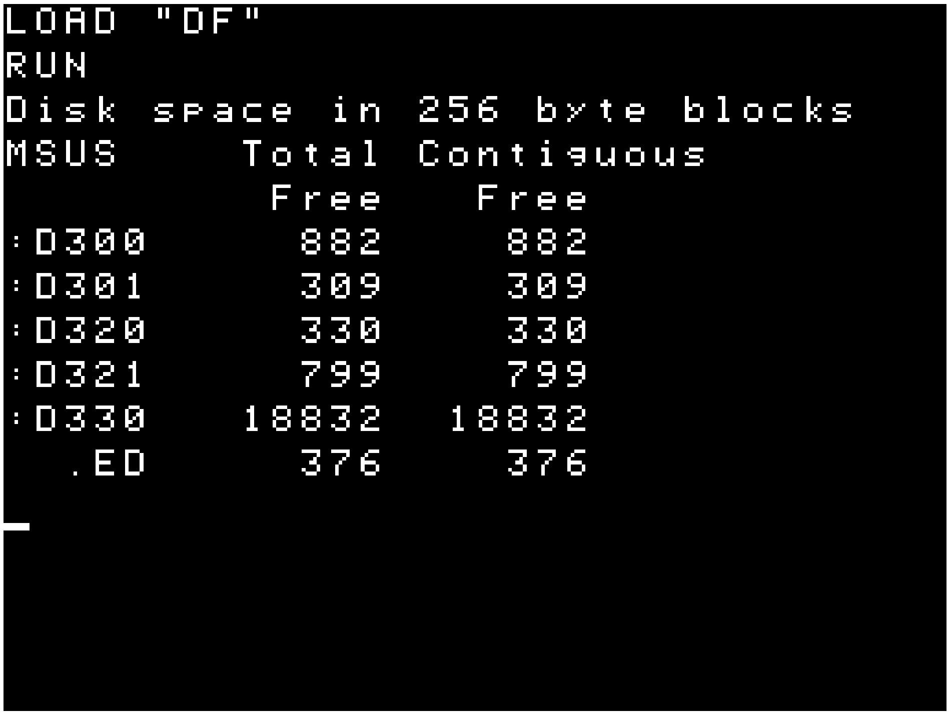
The DATE Program

The DF program shows the amount of total unused disk space and the largest unused space. (in a PACKed disk, these are the same). This is an example of a program that needs to be edited depending on your system. On an HP85B, which always has at least a small EDISK, no editing is needed. On all other systems, if there is no EDISK, line 110 must be changed as shown in the third image below.



```
LOAD "DF"
LIST 20,20
20 REM ON LINE 110 CHANGE THE 6
TO A 5 IF NO EDISK
LIST 110,110
110 DATA 6,":D300",":D301"
```

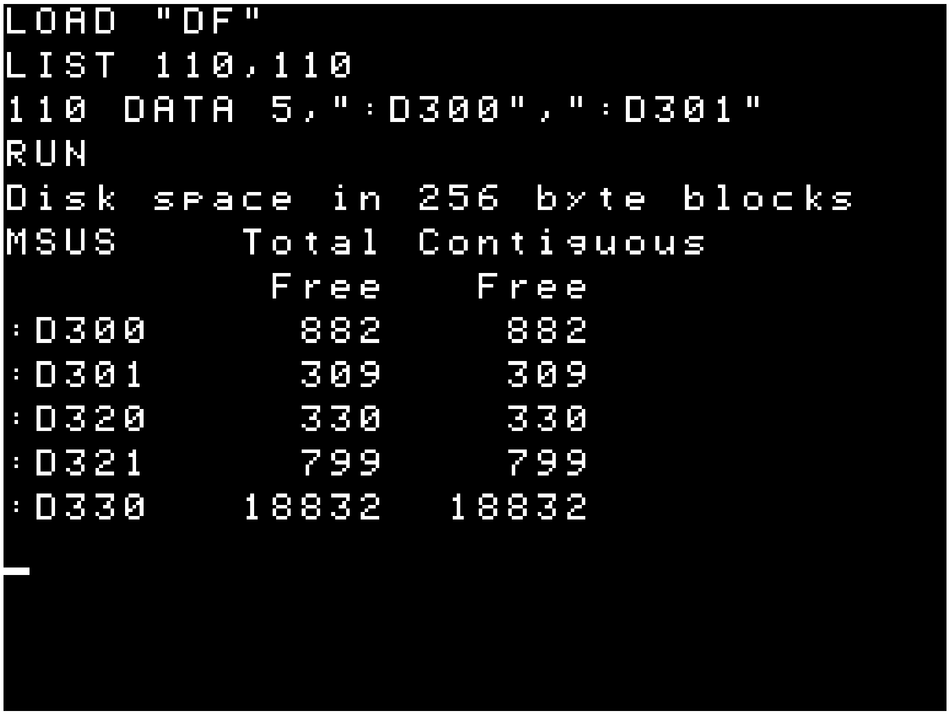
The DF Program (Disk free space)



```
LOAD "DF"
RUN
Disk space in 256 byte blocks
MSUS     Total Contiguous
          Free    Free
:D300      882     882
:D301      309     309
:D320      330     330
:D321      799     799
:D330    18832   18832
  .ED      376     376
```

Running DF on an HP85B

(Cont'd)



```
LOAD "DF"
LIST 110,110
110 DATA 5,":D300",":D301"
RUN
Disk space in 256 byte blocks
MSUS     Total Contiguous
          Free    Free
:D300      882     882
:D301      309     309
:D320      330     330
:D321      799     799
:D330    18832   18832

```

Running DF on an HP85A, after editing of line 110

The SINE program is as boring as you would expect it to be, and is not worthy of a screen capture.

The programs are used to test the two RGB LEDs on the back edge of EBTKS.
You can turn the LEDs off with SETLED 3,0,0,0

```
LED
LEDTEST1
LEDTEST2
```

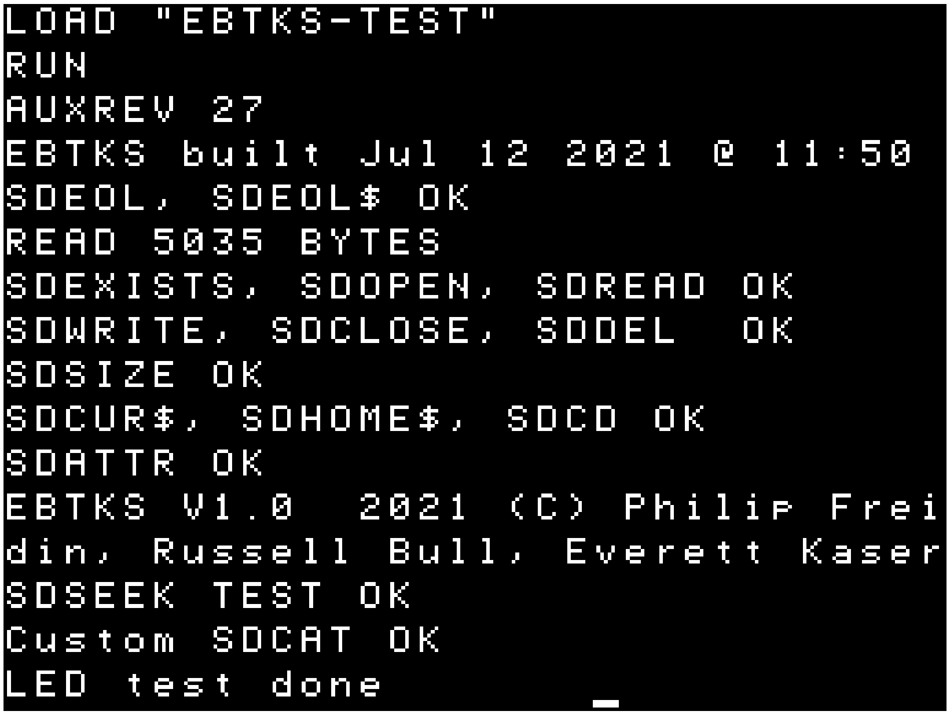
Two disk speed benchmarking programs

```
RWBENCH85A
RWBENCH85B  (this can also be used on HP86 and HP87)
```

were used to get the performance data presented at the end of this forum article . The program will probably need editing depending on whether you have EDISK enabled, and whether you have an HP9121 floppy disk drive attached via an HPIB interface with a msus of 700

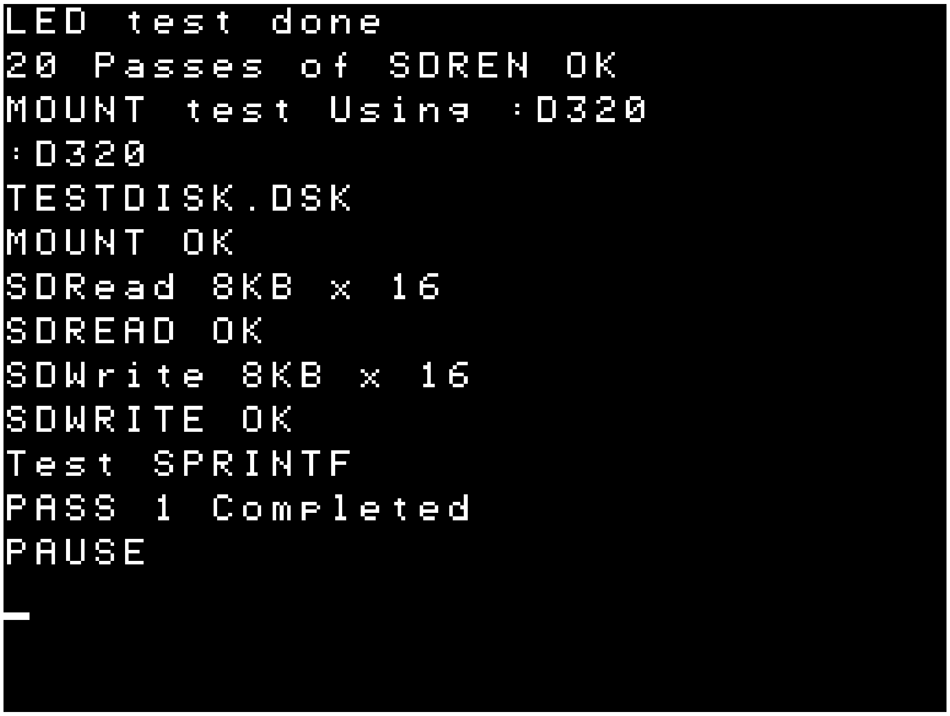
The **EBTKS-TEST** program checks multiple parts of EBTKS, and does not need any editing. On an HP85B it takes about 45 seconds to run. Here are the expected two screen pages:

(Cont'd)



```
LOAD "EBTKS-TEST"
RUN
AUXREV 27
EBTKS built Jul 12 2021 @ 11:50
SDEOL, SDEOL$ OK
READ 5035 BYTES
SDEXISTS, SDOPEN, SDREAD OK
SDWRITE, SDCLOSE, SDDEL  OK
SDSIZE OK
SDCUR$, SDHOME$, SDCD OK
SDATTR OK
EBTKS V1.0  2021 (C) Philip Frei
din, Russell Bull, Everett Kaser
SDSEEK TEST OK
Custom SDCAT OK
LED test done
```

Running EBTKS-TEST on an HP85B, page 1



```
LED test done
20 Passes of SDREN OK
MOUNT test Using :D320
:D320
TESTDISK.DSK
MOUNT OK
SDRead 8KB x 16
SDREAD OK
SDWrite 8KB x 16
SDWRITE OK
Test SPRINTF
PASS 1 Completed
PAUSE
```

Running EBTKS-TEST on an HP85B, page 2

Pressing CONT will repeat the test and increment the PASS count.

What else is on the MicroSD Card

The Directory Tree diagram near the top of this page shows many directories.
This is documented in detail here A Guided tour of the MicroSD Card

Here is a brief summary.

root	covered above
BAS	BASIC programs in ASCII source form. SDSAVE /BAS/DF.BAS stores the DF program in source form SDGET /BAS/DF.BAS loads the source file DF.BAS If these .BAS files are transferred to a PC, they can be edited with a text editor
disks	This is where the disk image files reside
EK_Disks	This is a copy of all the disk images found in Everett Kaser's Series80 emulator
help85	All the help text for HP85A/B are in this subtree. This is still a work in progress
help87	All the help text for HP86A/B and 87/87XM are in this subtree. This is still a work in progress
Original_images	Disk and tape images of blank media. Used when creating a new Disk or Tape
Printers	Output that would normally go to the printer can be redirected to a file in this directory
roms85	All of the available ROMs compatible with HP85A/B
roms87	All of the available ROMs compatible with HP86A/B and HP87/87XM
tapes	This is where the tape image files reside. unused on HP86A/B and 87/87XM
testfiles	A test directory used during development, and by EBTKS-TEST
TOK	BASIC programs in tokenized form. Functionally the same as saved programs in the emulated disks. SDLOAD /TOK/DF.TOK loads the DF program. SDSTORE /TOK/DF.TOK saves the DF program in tokenized form.

EBTKS Configuration

The **checklist page** that is included with your EBTKS lists the initial settings for your EBTKS. These are included in the CONFIG.TXT file that is in the root directory of the provided SD Card.
See Working with CONFIG.TXT for details of how to make changes to this file.
The checklist also lists the expected configuration of your system, with regard to the following:

- Is the tape drive connected (for HP85A and HP85B)
- It is expected that there is no ROM module or other ROM card installed as EBTKS should be able to provide all the ROMs you need
- On HP85A, don't install an extra 16 KB memory module as EBTKS can provide this memory.
- On HP85B, EMC memory modules can be used for EDisk. See EMC for details.

(Cont'd)

- Typically, a HP85B has 3 option ROMs pre-installed on the main board:

    - I/O (Octal 300)
    - Mass Storage (Octal 320)
    - EDisk (Octal 321)

  Do not enable these ROMs on EBTKS on an HP85B system, as it will cause incorrect operation. A similar constraint exists for HP86A/B and HP87/87XM pre-installed ROMs.

- EBTKS uses module select code 3 for an emulated HPIB interface that is the gateway to emulated floppy disk drives and emulated hard disk drives. It is also used for an emulated printer Therefore, if you have a real HPIB interface module plugged in for

    - Other real disk drives
    - Test and Measurement equipment
    - Data acquisition
    - Printers
    - Plotters
    - Emulated disks that connect over real HPIB

  the real HPIB interface must not use select code 3. By default, HP ships HPIB interface modules with select code 7.

As you will see on the Working with CONFIG.TXT page you can make edits to the file by putting the SD Card into a PC and using a simple text editor like notepad.

This allows changing the initial disk and tape drive assignments to various LIF image files. Even though it looks like you can add new additional sections following the existing format, please don't, as things will probably break. At a later time, hopefully an EBTKS owner will write a utility program that runs on the HP85 and automates simple changes to the CONFIG.TXT file.

Note:

> CONFIG.TXT is an ASCII text file in JSON format. Editors like WordPad and MSWord add additional information like character height, font selection, and other formatting control. This will corrupt the file.
> While Notepad can be used for editing CONFIG.TXT, another far superior free editor that does not add formatting is Notepad++ which can be found with Google Search.
> For Linux/Unix systems, suitable editors include VI, VIM, EMACS, Nano, Atom, Gedit.

A few New Keywords

The following are useful initial keywords to check the firmware and AUXROM versions, and which ROMs have been loaded. These are all just typed at the BASIC prompt. See AUXROM Keywords for the full list of over 57 new keywords, with more detailed explanations and examples

(Cont'd)

AUXREV

Returns the AUX ROMs revision# (from ROM 361)

EBTKSREV$

Returns a string containing date and time of building the EBTKS firmware

LISTROMS 0

Displays all the possible ROMs for HP85/86/87 and their Octal/Decimal ID number

LISTROMS 1

Displays currently installed ROMs and their Octal/Decimal ID number. AUXROMS are loaded as a set with Octal IDs 361, 362, 363, 364. LISTROMS will only list the first one

Links

EBTKS Downloads

Updating the EBTKS Firmware

AUXROM Keywords

Connecting the serial diagnostic port

Highlights

There are several help pages that list the available commands. Just type the number 0 to 6 and enter to get a short help page.

The serial diagnostic port Help

This section of EBTKS is likely to change since it is primarily a debug tool for the EBTKS developers

Here are the 7 pages of help

```
EBTKS> 0

EBTKS Control commands - not case-sensitive

0     Help for the help levels
1     Help for Display Information
2     Help for Diagnostic commands
3     Help for Directory and Time/Date Commands
5     Help for Developers
6     Help for Demo
```

(Cont'd)

```
EBTKS> 1
Commands to Display Information
show -----    Show commands have a parameter after exactly 1 space
     log      Show the System Logfile
     boot     Show the messages from the boot process, sent to Serial port
     CRTboot  Show the messages sent to the CRT at startup
     config   Show the CONFIG.TXT file
     media    Show the Disk and Tape assignments
     mb       Display current mailboxes and related data
     CRTVis   Show what is visible on the CRT
     CRTAll   Show all of the CRT ALPHA memory
     key85_O  Display HP85 Special Keys in Octal
     key85_D  Display HP85 Special Keys in Decimal
     key87_O  Display HP87 Special Keys in Octal
     key87_D  Display HP87 Special Keys in Decimal
     other    Anything else is a file name path
```

```
EBTKS> 2
Commands for Diagnostic
la setup      Set up the logic analyzer
la go         Start the logic analyzer
addr          Instantly show where HP85 is executing
kbdcode       Show key codes for next 10 characters in the keyboard byffer
clean log     Clean the Logfile on the SD Card
sdreadtimer   Test Reading with different start positions
SDCID         Display the CID information for the SD Card
PSRAMTest     Test the 8 MB PSRAM. You probably should do the PWO command when test has finished
ESP32 Prog    Activate a passthrough serial path to program the ESP32
pwo           Pulse PWO, resetting HP85 and EBTKS
```

```
EBTKS> 3
Directory and Date/Time Commands
dir tapes     Directory of available tapes
dir disks     Directory of available disks
dir roms      Directory of available ROMs
dir root      Directory of available ROMs
Date          Show current Date and Time
SetDate       Set the Date in MM/DD/YYYY format
SetTime       Set the Time in HH:MM 24 hour format
adj min       The U and D command will adjust minutes
adj hour      The U and D command will adjust houres
U             Increment the time by 1 minute or hour
D             Decrement the time by 1 minute or hour
```

```
EBTKS> 4
Commands for Auxiliary programs
```

```
EBTKS> 5
Commands for Developers (mostly Philip)
crt 1         Try and understand CRT Busy status timing
crt 2         Fast CRT Write Experiments
crt 3         Normal CRT Write Experiments
```

(Cont'd)

```
crt 4         Test screen Save and Restore
crt 5         Test writing text to HP86/87 CRT
```

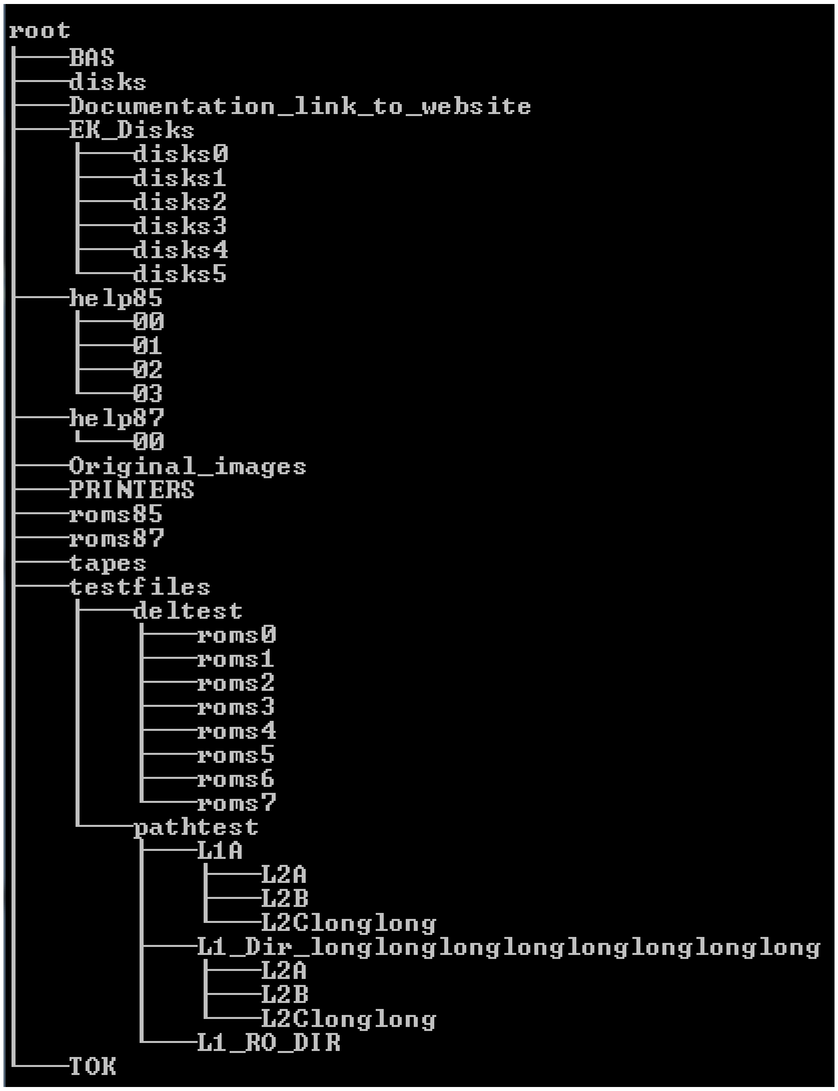
```
EBTKS> 6
Commands for Demo
graphics test  Set graphics mode first
jay pi         Jay's Pi calculator running on Teensy
```

Boot logging

Every time EBTKS starts up it creates three logs.

- Short list of ROM and Disk Mounts can be sent to the CRT. Can be disabled in CONFIG.TXT from being displayed
- A detailed report on the processing of the CONFIG.TXT file. Always sent to the serial diagnostic port.
- A similar report to the report that is sent to the serial diagnostic port, is appended to a log file on the MicroSD Card

A Guided tour of the MicroSDCard



```
root
├───BAS
├───disks
├───Documentation_link_to_website
├───EK_Disks
│   ├───disks0
│   ├───disks1
│   ├───disks2
│   ├───disks3
│   ├───disks4
│   └───disks5
├───help85
│   ├───00
│   ├───01
│   ├───02
│   └───03
├───help87
│   └───00
├───Original_images
├───PRINTERS
├───roms85
├───roms87
├───tapes
├───testfiles
│   ├───deltest
│   │   ├───roms0
│   │   ├───roms1
│   │   ├───roms2
│   │   ├───roms3
│   │   ├───roms4
│   │   ├───roms5
│   │   ├───roms6
│   │   └───roms7
│   └───pathtest
│       ├───L1A
│       │   ├───L2A
│       │   ├───L2B
│       │   └───L2Clonglong
│       ├───L1_Dir_longlonglonglonglonglonglonglong
│       │   ├───L2A
│       │   ├───L2B
│       │   └───L2Clonglong
│       └───L1_RO_DIR
└───TOK
```

MicroSD Card Directory Tree

Content coming soon

For now, here is MicroSD Card Summary

Working with CONFIG.TXT

In the root directory of the SD Card there is a file named CONFIG.TXT . It contains all the configuration information for EBTKS. This file is read when the Series80 computer starts up, and all the configuration is completed before the Series80 computer presents its first prompt to the user. The file format is JSON, and as such, it depends on a very specific format for the various entries, and the way that sections are nested. At the time of writing this page of the documentation, the best way to make changes to CONFIG.TXT is to do the editing and format verification on you PC, as there is no convenient way to do it directly on the Series80 computer.

Before you get started

Top

As mentioned above, the CONFIG.TXT file is in JSON format. The EBTKS reader and parser expect the file to be syntactically correct, and since it is processed during system startup, if there are errors (a missing comma, non-matching curly-brace) the failure won't be reported, but your system won't start up correctly. Fortunately, there is a JSON file format checker available on-line that can be used to validate that any edits that you make to this file haven't broken any rules.

So, let's get comfortable with using the format checker, before we make any changes, so you will know what the results should be if you have made a mistake, or if you did your edits correctly.

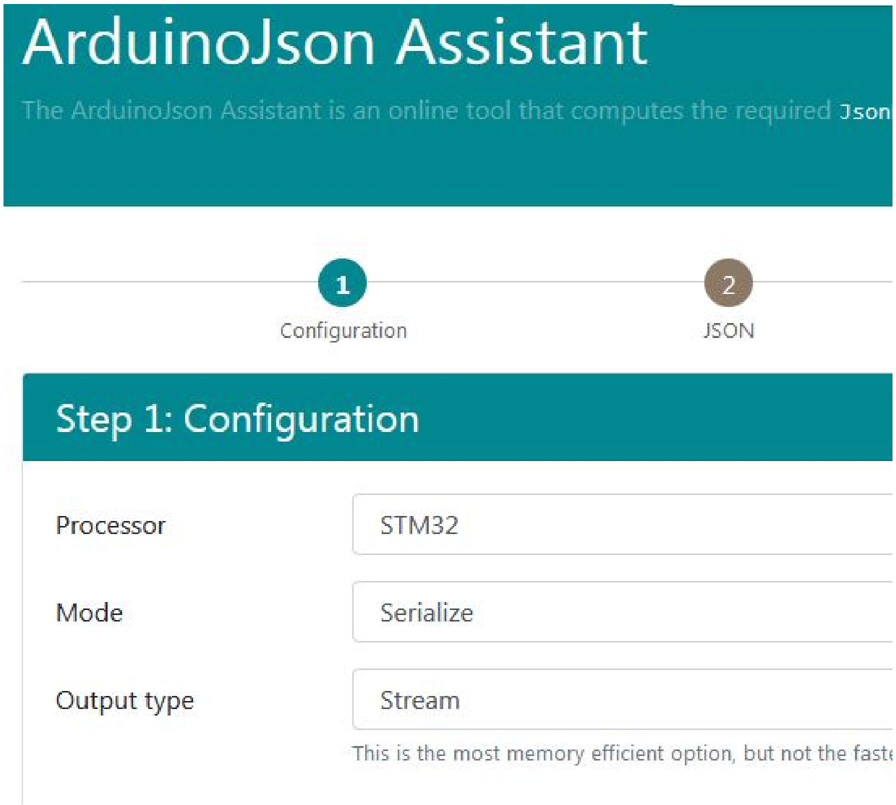
1. Copy the CONFIG.TXT file from the SD Card to your desktop/laptop computer, and open the file in a text editor, like notepad (on a windows PC). There is also an example of CONFIG.TXT for a HP85A at the bottom of this page.

2. Open this link in a browser: ArduinoJson Assistant

3. Set the configuration as shown:

    - Processor: STM32
    - Mode: Serialize
    - Output type: Stream

(Cont'd)



ArduinoJson Assistant

The ArduinoJson Assistant is an online tool that computes the required Json

Step 1: Configuration

Processor: STM32

Mode: Serialize

Output type: Stream

This is the most memory efficient option, but not the faste

Then click the button labeled “Next: JSON”

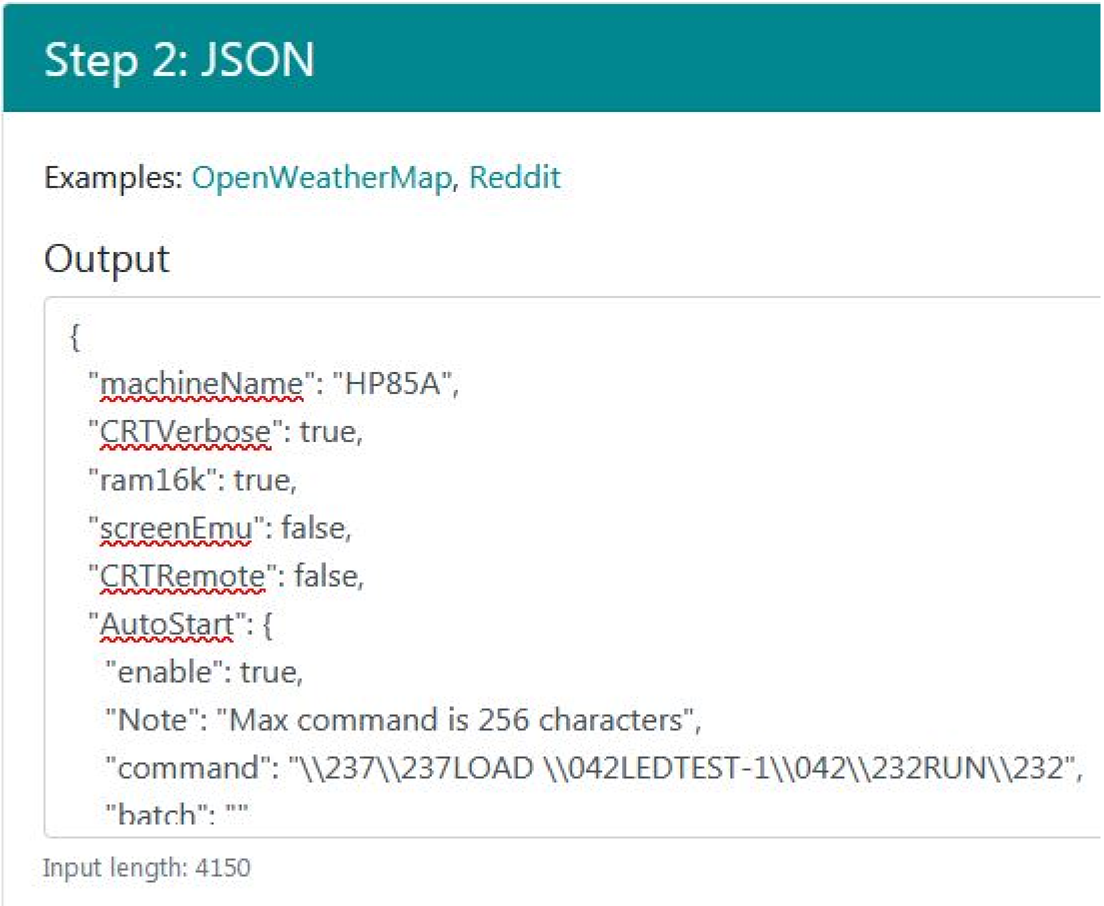
4. Select all the text in the text box labeled “Output” box, and delete it.
   (on a Windows PC, click inside the box, type Ctrl-A to select all, then the delete key)

5. In the text editor that is showing CONFIG.TXT, select all the text and copy it to the clipboard.
   (on a Windows PC, click anywhere on the text in the editor and type Ctrl-A to select all, then type Ctrl-C to copy it all to the clipboard)

6. Paste the clipboard contents into the ArduinoJson Assistant Output box.
   (on a Windows PC, click anywhere inside the box, and type Ctrl-V)

7. Scroll the “Output” box to the top, and it should look something like this:

(Cont'd)



Step 2: JSON

Examples: OpenWeatherMap, Reddit

Output

```
{
  "machineName": "HP85A",
  "CRTVerbose": true,
  "ram16k": true,
  "screenEmu": false,
  "CRTRemote": false,
  "AutoStart": {
   "enable": true,
   "Note": "Max command is 256 characters",
   "command": "\\237\\237LOAD \\042LEDTEST-1\\042\\232RUN\\232",
   "batch": ""
```

Input length: 4150

Don’t worry about the squiggly red lines under some of the text. On my browser this is just automatic spell checking getting confused. You may or may not see this depending on your operating system and browser. It does not indicate an error.
In particular, note that at the bottom left corner of the Output box, it shows “Input length: 4150” and the text is black/gray. This indicates that there are no errors in the JSON formatting.

8. Now delete one of the commas at the end of a line, or a quote character, or a colon.
   You should instantly see the message at the bottom left corner of the Output box turn red, and indicate that there is an error, and where the error has occurred. Restore the deleted character and the Input length message should return.

As just demonstrated, you can edit the CONFIG.TXT directly within the ArduinoJson Assistant window, after you copy CONFIG.TXT to that window. This will give you continuous live updates of whether the text is JSON syntax valid. When you are done editing (and the syntax has been validated), you can select all the text with Ctrl-A , copy to clipboard Ctrl-C, and then paste it back into notepad (or whatever) with Ctrl-V. Then save the updated CONFIG.TXT with Ctrl-S. The updated CONFIG.TXT should then be copied back to the root directory of the SD Card

When making any edits to CONFIG.TXT you must at a minimum do the above test after you have completed your edits. While getting the “Input length” message indicates there are no

(Cont'd)

syntax errors in your CONFIG.TXT file, you still need to be very careful of all file pathnames, and spelling of parameter values, since the ArduinoJson Assistant will not catch those types of errors. All text values must be quoted. The *true* and *false* parameters values must not be quoted, and they are checked by ArduinoJson Assistant.

OK, now let’s look at the various sections of CONFIG.TXT

System Settings

Top

```
1    "machineName": "HP85A",
2    "CRTVerbose": true,
3    "ram16k": true,
4    "screenEmu": true,
5    "CRTRemote": true,
```

The first line (“machineName”) specifies the Series80 model number. The EBTKS firmware uses this information to enable certain features and to check some of the setting specified in the CONFIG.TXT file

“HP83”	“HP9915A”	“HP85A”
“HP85AEMC”	“HP85B”	“HP9915B”
“HP86A”	“HP86B”	“HP87”
“HP87XM”		

Line	Explanation
2	If true, the extended Boot Message is sent to the Series80 CRT. This includes which ROMs are active, and the associations between msus$ and LIF Files
3	For the HP85A only, EBTKS can provide 16 KB of RAM if this parameter is set true.
4	Under development, always set true at this time
5	Under development, always set true For HP85A/B and false for HP86/87

Top

Tape Drive

```
1 "tape": {
2   "enable": true,
3   "filepath": "/tapes/tape1.tap"
4 },
```

Tape Emulation Configuration

Line	Explanation
1	This is the configuration section for the tape drive emulation.
2	Controls whether the section is enabled, which is only allowed on HP85A and HP85B. The whole section has no effect if this parameter is set to false. If this section is enabled, the real tape drive in the HP85A or HP85B must be disabled, which can be done by unplugging the two Flat-Flex cables to the tape drive.
3	Specifies the directory path and file name for the LIF Image file that represents an emulated tape.
4	This is the end of the configuration section. A trailing comma is required if this section is followed by another section at the same logical indent level

ROMs Section

Top

EBTKS can support up to 20 Option ROMs, four of which are the AUXROMs, which must always be present. The rest are optional. Option ROM 000 is built into all Series80 computers. All other option ROMs are bank- switched into the same address space, so only one can be active at any time. It is likely that some will always be required for the HP85A if you want to use the Disk Drive emulation (rom320B and rom321B), and some should not be enabled because they are pre-installed in the computer (HP85B usually has 3 Option ROMs pre-installed). The ROMs section of the CONFIG.TXT file is a nested structure. The outer part specifies that it is the ROM section, and the default directory on the SD Card where all the ROM images are stored. The inner part of the ROM section starts with [and ends with]. Here is the outer section, which you will probably never need to change.

```
1 "optionRoms": {
2   "directory": "/roms85/",
3   "roms": [
4
5    Multiple ROM entries (see next section), each of which
6    ends with a comma except the last ROM entry
7
8   ]
9 },
```

(Cont'd)

ROM Configuration, outer part

Line	Explanation
1	This is the configuration section for the Option ROMs.
2	The default directory for the ROM images is /roms85/ On an HP86 or HP87 this would be /roms87/ The directory string must include the leading and trailing slash character /
3	Start of the inner section
5	The inner sections go here, each is 6 lines long, and all end with a comma except the last one. See the next section
8	End of the inner section
9	End of the Option ROMs section. A trailing comma is required if this section is followed by another section at the same logical indent level

ROM Entries

Top

For each Option ROM, there is an entry within the ROM Section, described above.
If the entry is disabled (enable set to false), then it does not consume any memory in EBTKS. Enabled ROMs each consume 8k bytes, from a memory region that has pre-allocated room for 18 ROMs. Four of these must always be the AUXROMs. The remaining 14 can be used for any combination of Option ROMs. Another restriction on the number of ROMs that can be enabled is that for the HP85A/B there can be no more than 14 Option ROMs. For the HP86/87 it is 15 Option ROMs. When counting ROMs, the built in ROMs must be include, and although EBTKS requires 4 AUXROM ROMs, they only count as 1 in this calculation of the system bases restriction (14 or 15) but they do count as 4 ROMs with regard to the allocated memory for 18 total. If you do the math, you will see that room for 18 ROMs is just enough.

To summarize the above:

- For an HP85A ROMs 000 and the AUXROMs count as 2 total, allowing for up to 12 additional Option ROMs.
- For an HP85B ROMs 000, 300 (I/O), 320(Mass Storage), 321(EDisk) pre installed and AUXROMs counts as 5, allowing for up to 9 additional Option ROMs.
- For an HP87 ROMs 000, 001 (87 Graphics), 320(Mass Storage) pre installed and AUXROMs counts as 4, allowing for up to 10 additional Option ROMs.
- For an HP87XM ROMs 000, 001 (87 Graphics), — needs to be checked —

In general, you should only enable the ROMs you need, as enabling ROMs you don’t need will slow down some of the Series80 computer functions. Another very important rule is that you must

(Cont'd)

not enable two copies of the same ROM. This could happen if your Series80 has a ROM draw, the EPROM module or PRM-85 plugged in (which should not be needed, since EBTKS can provide the ROMs), or if they are pre- installed on the computer's main board, which is the case for all Series80 computers except the HP85A.

The order of the ROM entries does not matter.

If you are not sure which ROMs are pre-installed on your Series80 computer, edit the CONFIG.TXT file so that all the ROMs are disabled except for the four AUXROMs. Then start your Series80 computer and enter the command

LISTROMS 1

and it will list the installed ROMs.
Note: For the AUXROMs, it will only list the first AUXROM, Octal 361.

Example ROM Entry

```
1 {
2   "Note": "For 85B floppies and 5, 10 MB hard disk. Use with rom321B, can be used on 85A",
3   "description": "85B Mass Storage",
4   "filename": "rom320B",
5   "enable": true
6 },
```

ROM Entry, inner part

Line	Explanation
1	Start of a ROM entry
2	Documentation text about the entry, so you don't have to remember what each ROM number means
3	The functional name of the ROM
4	File name for the ROM image. This is appended to the directory specified in the outer section. See line 2 in the previous section. These files must be exactly 8192 bytes in length
5	Whether to enable the ROM. Either true or false, without quotes. Note that there is no trailing comma, as this is the last line at this level within this block
6	End of a ROM entry

Top

Disk Drives

EBTKS can support multiple emulated disk drives. Currently two types are supported:

- Floppy disks with 1040 data blocks (266,240 bytes, usually just referred to as 256 KB) not including the directory area
- Hard disks with 18832 data blocks (4,820,992 bytes, usually referred to as 5 MB) notincluding the directory area

The disk emulation is dependent on a limited functionality HPIB emulation that is also implemented by EBTKS. In particular the HPIB emulation only provides sufficient functionality for the disk and printer emulation, but does not implement a physical HPIB, and there is no HPIB physical connector for you to connect other HPIB devices. The emulated HPIB behaves like an HPIB controller (HP82937A), and so standard HP software Option ROMs like the Mass Storage ROM (320 Octal) and the I/O ROM (300 Octal) can communicate with it, and can't tell that there is no physical HP82937A implemented by EBTKS.

By default the real HP82937A is shipped with a select code of 7, and EBTKS is designed to co-exist with one being plugged in. You might want to do this if you have real disk drives or you may be using HPIB to control electronic test equipment or data acquisition products.

To avoid conflict, by default the EBTKS emulated HPIB uses select code 3.

Understanding the msus$ strings

Top

HP disk drives that connect to the Series 80 computers via HPIB are organized into three levels:

- HPIB Select Code
  - Device number on the selected HPIB Select Code
    - Drive Drive number that is contained in the specified Device number

Physically this is implemented as follows:

- The HPIB Select Code is associated with a specific HPIB interface plugged into the back of the Series80 computer
- The Device is a box with power supply and one or disk drives that connect to the HPIB interface via a HPIB cable
- Within the Device box there are one or more Disk drives, usually numbered as Unit 0, 1, …

In Series80 BASIC, specific drives are referenced with a msus$ string. The format is ":Dsdu", where s, d, and u are

s - The Select Code

d - The Device Number

u - The Unit Number

(Cont'd)

Therefore, ":D300 refers to HPIB Select code 3 (EBTKS default), Device box 0, Unit 0. In EBTKS, all this is emulated, but the end result is that you use the same msus$ string as you would with physical HPIB interface, cable, disk drive box, and the disk drives within it.

Disk Drives Configuration

Top

Just as the msus$ implements a 3 level hierarchy to get to the specific disk drive unit, the CONFIG.TXT file similarly uses a 3 level JSON structure to achieve the same thing. There is a 1-to-1 mapping between the msus$ and the layout of this section of CONFIG.TXT

Disk Drives, Level One

Top

The first level specifies the emulated HPIB interface, and it only has one parameter, the select code.

```
1  "hpib": {
2    "select": 3,
3    "devices": [
4
5      Multiple Device sections, each start with { and end with }
6      There is a comma after each } , except for the last one.
7
8    ]
9  }
```

Disk Drives, Level One

Line	Explanation
1	Start of the HPIB entry
2	Specifies the Select Code for the Emulated HPIB interface
3	The start of an array (list) of devices, indicated by the [character
5	The Devices list (equivalent to a Disk Drive case with power supply and one or more drives) is made up of one or more sections, each starting with { and ending with }
6	Without showing the content, an example array would be {}, {}, {}, {} . Note that there is no trailing comma after the last Device section
8	The closing] indicates the end of the array of devices
9	End of a HPIB entry

(Cont'd)

Disk Drives, Level Two

Top

The second level is an array (list) of Devices (equivalent to a disk drive box that has one or more disk drives). As described in level one (above), each entry starts with a { and ends with }. If an entry is followed by another, there is a separating comma.

```
 1  {
 2    "Comment": "All blocks must have different device numbers",
 3    "type": 0,
 4    "device": 0,
 5    "enable": true,
 6    "drives": [
 7
 8      Multiple Disk Drive sections, each start with { and end with }
 9      There is a comma after each } , except for the last one.
10
11    ]
12  },
```

Disk Drives, Level Two

Line	Explanation
1	Start of a Device entry
2	A reminder that device numbers need to be unique.
3	Specifies the type of Disk drives being emulated. All the Drives for a given Device must be the same type. Type 0 is a 256 KB floppy, type 4 is a 5 MB hard drive (all emulated, no hardware other than EBTKS)
4	The Device number that must be unique. Can be 0 through 7
5	The Device can be enabled/disabled. If disabled, it will disable all the Disk Drives for this Device. Equivalent to turning the power off to the Disk Drive box
6	The start of an array (list) of Disk Drives, indicated by the [character
8	The Disk Drive array is a list of the Disk Drives within a Device (Disk Drive Box). Each Disk Drive is specified with a block that starts with { and ends with }
9	Without showing the content, an example array would be {}, {}, {}, {} . Note that there is no trailing comma after the last Disk Drive section
11	The closing] indicates the end of the array of Disk Drives
12	End of a Device entry

(Cont'd)

Disk Drives, Level Three

Top

The third level is an array (list) of Disk Drives (that are within a Disk Drive Box). As described in level two (above), each entry starts with a { and ends with }. If an entry is followed by another, there is a separating comma. So, at last we get to the Disk Drives (emulated)

```
1 {
2   "Comment": "msus$ 300",
3   "unit": 0,
4   "filepath": "/disks/BETAUTIL.DSK",
5   "writeProtect": false,
6   "enable": true
7 }
```

Disk Drives, Level Three

Line	Explanation
1	Start of a Disk Drive entry
2	Document what the msus$ is for this Disk Drive instance. The first digit is the HPIB Select code, the second digit is the Device Number, and the third digit is this specific Disk Drive within the specified Device. Within a specific Device, the Disk drive numbers must be unique, and are between 0 and 3
3	The Disk Drive Number, 0 through 3
4	Each Disk Drive must be associated with a file on the SD Card that holds the data of the emulated Disk that is currently loaded into a Disk Drive. If the Drive is a Floppy Disk Drive for example, the specified file represents a Floppy Disk. This line specifies the directory path and file name for the LIF Image file that represents an emulated disk. File names and paths are not case sensitive
5	WriteProtect is a place holder at the time of writing this documentation. It may be implemented some time in the future
6	Each Disk Drive can be individually enabled. If it is disabled (set this parameter to false), the Disk Drive cannot be enabled until the CONFIG.TXT file is modified and the Series80 computer is rebooted. MOUNTing a LIF file on a disabled Disk Drive will not work. (equivalent to putting a floppy disk in a drive that has the power off)
7	End of a Disk Drive entry

Top

(Cont'd)

Disk Drives, putting it all together

Without all the details described above, here is the structure overview of the Disk Drive section of CONFIG.TXT that is listed at the bottom of this page. The indent levels directly relate to the above descriptions.

The description specifies 4 floppy drives with msus$ of 300, 301, 320, 321. Two 5 MB hard drives, but only one is enabled with msus$ of 330, and finally an emulated printer with device number 310 (which is not a msus$). The printer section will be discussed in the next section, but it is referred to here because it is emulated as a device connected to the emulated HPIB, and so appears in the same section as the Disk Drives.

If your browser window is narrow, the text below may be clipped on the right side. If this happens, there is a scroll bar at the bottom of the page that you can adjust to see all the text

```
"hpib": {
  "select": 3,
  "devices": [
    {  Device 0, Type 0 (Floppies), LIFs are in /disks/
      [ An Array of Floppy Disk Drives
        {Floppy drive 0, msus$ is 300,
                      LIF file is /disks/BETAUTIL.DSK} ,
        {Floppy drive 1, msus$ is 301,
                      LIF file is /disks/85Games1.dsk}
      ]   end of an array of floppies
    },
    {  Device 2, Type 0 (Floppies), LIFs are in /disks/
      [ An Array of Floppy Disk Drives
        {Floppy drive 0, msus$ is 320,
                      LIF file is /disks/85Games2.dsk} ,
        {Floppy drive 1, msus$ is 321,
                      LIF file is /disks/85GamesMisc.dsk}
      ]   end of an array of floppies
    },
    {  Device 3, Type 4 (5 MB Hard drive), LIFs are in /disks/
      [ An Array of 5 MB Hard Disk Drives
        {5 MB Hard Disk drive 0, msus$ is 330,
                      LIF file is /disks/5MB_scr.dsk} ,
        {5 MB Hard Disk drive 1, msus$ is 331,
                      LIF file is -none- because not enabled}
      ]   end of an array of hard drives
    },
    {
    the HPIB printer is also a device on the emulated
    HPIB bus. See the next section
    }
  ]   end of array of devices
}   end of hpib
```

Printers

EBTKS can support multiple emulated HPIB printers, that take the data that is output to the printer and redirects it to be written to a file on the SD Card. This should not be interpreted as multiple models of HP printers, just multiple instances of very simple text only printers. The emulated printer(s) appear as an HPIB device, and so the configuration is included in the HPIB part of the CONFIG.TXT file, after the disk drive sections. Given the default HPIB select code of 3, the emulated printer has a default device number of 10, and thus the full device number is 310. This should not be confused with a msus$ of 310, but is the reason that in the disk drive section above, we skipped device 1, and used device 0, 2, and 3 to avoid possible confusion.

To use this printer redirection capability, the Printer/Plotter ROM must be enabled.

To activate the redirection of PRINT statements from sending text to the built-in printer, run this command:

PRINTER IS 310

After executing this statement, printer output is appended to the designated file. To return to output going to the built in printer, run this command:

PRINTER IS 2

```
1  {
2    "Comment": "Device 10 on HPIB select code, i.e. 310",
3    "printer": "",
4    "device": 10,
5    "enable": true,
6    "filepath": "/printers/printfile.txt"
7  }
```

Disk Drives, Level Three

Line	Explanation
1	Start of the Printer entry
2	Document what the Device ID is for the Printer. The first digit is the HPIB Select code, the remaining two digits are the device id on that (emulated) HPIB bus
3	This line identifies this section as an emulated printer section, rather than a Disk Drive section.
4	The emulated Printer device number can be 0 to 31, but I recommend 10 to 31 to avoid confusion with Disk msus$
5	The emulated Printer can be individually enabled. If it is disabled (set this parameter to false), directing print data to the SD Card will not be available.
6	The emulated Printer must be associated with a file on the SD Card that receives the printed text. This line specifies the directory path and file name for that file. File names and paths are not case sensitive .
7	End of a Printer entry

AutoStart

Top

For HP85A and HP85B (Series80 computers with Tape Drives), if a tape cartridge is already in the tape drive when power is turned on, and if the tape contains a program named “Autost” (this is case sensitive), the program will be loaded from the tape and run. If the program is not near the beginning of the tape, this may take quite a while. Even if there is no “Autost” program on the tape (including an EBTKS emulated tape and tape drive), the directory needs to be read, and with that, the CRT is turned off while this occurs (standard behavior of HP85A/B to save power) just as the system is starting up. EBTKS supports this on the emulated Tape drive, but also provides some enhanced capabilities.

Autostart from Tape

Top

To enable the standard Autost program from the emulated tape drive, the parameters “enable” and “enableTapeAutostart” should both be set to true, “command” and “batch” should both be empty strings.

```
"AutoStart": {
    "enable": true,
    "enableTapeAutostart": true,
    "Note": "Max command is 256 characters",
    "command": "",
    "batch": ""
  }
```

Automatically enter commands from CONFIG.TXT

Top

EBTKS supports initial commands copied from the CONFIG.TXT file onto the CRT and then run. This is done by pretending to be characters coming from the keyboard as soon as the Series80

(Cont'd)

computer has completed booting up. You will actually see the commands appearing on the CRT when this is done. The command string in CONFIG.TXT can be up to 256 characters, and must be on a single line. All of the special keys on the keyboard can be entered by using their 3 digit octal value preceded by a double back slash.
Here is a link to the Octal codes . When entering octal codes in the “command”: parameter, each octal code **must be preceded by two back slash** characters. Since the ENDLINE character can used in the string, multi-line commands are possible.

Some common codes to build a command string

Batch files have different rules, see section below

Desired Key	Code to use
CLEAR	(Clear Screen) `\\222`
-LINE	(clear to end of line) `\\240`
`"` (double quote)	`\\042` or `\"`
ENDLINE	`\\232`
RUN	`\\215`
DOWNCURSOR	`\\242`
\ (backslash)	`\\134`

With EBTKS installed, depending on the setting of CRTVerbose, the CRT may be displaying many lines of text or a clear screen. As you know, when you type a command, if there are characters on the line after the command you have typed, it will mess up your command. For HP85A/B, you must also make sure that the previous line does not have a character in the last position (character position 32) as that will cause the previous line to be treated as part of the command too. There are a few strategies that could work here:

A. Clear the screen before entering commands using octal `\\222` at the beginning of the command.
B. Clear the current line with “-line” code `\\240`, then move down 1 line (`\\242`) and clear that line as well (`\\240`), then enter the desired command. So the beginning of “command” would be
“`\\240\\242\\240` rest of the command”
This sequence assumes the cursor is in column 1, which is true at system start.
C. Roll the screen down two lines, since we know at initial start, the prior two lines (CRT buffer lines 62 and 63) will be blank the beginning of “command” would be `\\237\\237` as seen in the example.

(Cont'd)

To use this version of Autostart, let's look at an example that uses the third option:

```
"AutoStart": {
    "enable": true,
    "enableTapeAutostart": false,
    "Note": "Max command is 256 characters",
    "command": "\\237\\237LOAD \\042LEDTEST2\\042\\232RUN\\232",
    "batch": ""
  }
```

The rest of the example is made up of the following pieces

- LOAD which is the standard load command
- `\\042` which is the double quote character at the start of a file name for the load command
- LEDTEST2 which is the name of the program to be loaded
- `\\042` which is the double quote character at the end of a file name for the load command
- `\\232` which is the endline key, thus finishing the LOAD command, and causing the program to be read from the default storage device and placed into memory. With default settings of EBTKS, this would fetch the LEDTEST2 program from ":D300"
- RUN followed by `\\232` will now run the program.

Using this form of autostart allows quite complex startup, limited by the requirement that the command string is no more 256 characters.

Automatically enter commands from a batchfile

Top

EBTKS supports initial commands from a batch file that is specified in the CONFIG.TXT file. As detailed in the previous section, you must be aware of what might already be on the screen that could interfere with the command that are in the batch file. The same 3 strategies are available. There are three differences between the batch file approach to autostart and the single string described in the previous section.

1. You are not limited to 256 characters
2. Octal constants only have a single backslash \ preceding octal key codes
3. You don't need to use \232 for the end of line character, or \042 for the double quote characters. Normal ends of lines in the batch file will be interpreted as the end line that the HP85 expects.

To use this version of Autostart, let's look at an example:

```
"AutoStart": {
    "enable": true,
```

(Cont'd)

```
    "enableTapeAutostart": false,
    "Note": "Max command is 256 characters",
    "command": "",
    "batch": "autoexec.bat"
  }
```

The contents of the batch file could be:

```
LOAD "LEDTEST-1"
RUN
```

Unlike the batch files on Windows OS systems, this batch capability is very simple. Just the sequential transfer of text.

Extended Memory Control (EMC)

Top

EBTKS can provide up to 256 kB of Extended Memory for HP85B, HP86A/B, HP87/XM.
(256 kB at the time of writing this documentation. This may change in the future).
On the HP85B the only use for this memory is the Electronic Disk. Given the SD Card based disk emulation provided by EBTKS, this is probably of no practical use. On the HP86 and HP87 computers, this increases the available memory for program and data. Extended Memory is not support on the HP85A (unless a circuit modification is made which would make it compatible with the HP85B in this regard, but as for the HP85B, this would be of little utility).

Extended memory is measured in **Banks** which are 32 kB each, so 256 kB of memory is 8 Banks. You can enable any number of banks from 0 to 8. The **NumBanks** parameter sets the number of Banks of Extended Memory that EBTKS will provide.

Setting the starting bank number **StartBank** requires some minor calculation. Follow these steps:

StartBank for HP85B with no additional EMC modules

1. Set the EMC section of CONFIG.TXT in the root directory of the SD Card to the following

```
"EMC": {
  "enable": true,
  "NumBanks": 8,
  "StartBank": 3
}
```

(Cont'd)

StartBank for HP85B with additional EMC modules

1. Unplug EBTKS from your system and plug in the extended memory modules you plan to use with EBTKS (64 kB or 128 kB). Note: the 16K RAM module from HP is not compatible with HP85B computers.
2. The HP85B comes with 32kB of extended memory as Bank 2.
3. Start your calculation with an initial value of 3 and add 2 for each 64kB modules that you are installing, and add 4 for each 128kB modules that you are installing. The result is the calculated value. For example, if you plug in one 64kB module and one 128kB module, the calculation is 3+2+4 = 9
4. The calculated value of step 3 is the value to be used for **StartBank** parameter. Update the CONFIG.TXT file on the SD Card with this value, and plug the SD Card into the socket on the Teensy processor module.
5. EBTKS can now be plugged into your HP85B.
6. If you unplug any Extended memory modules, you will have to repeat this calculation. Well, actually, if you unplug a 128kB module, just subtract 4 from the previous calculated StartBank. If you unplug a 64kB module, subtract 2 from the previous calculated StartBank.
7. Confirm that you have the settings correct by doing a using the **DISC FREE A, B, ":D000"** command (see page 274 of the HP85B Owner's Manual). The reported available EDisc size is given in disk blocks. These are each 256 bytes. Take the value, and divide by 4 to get kB of EDisc. It should be close to the expected Extended memory provided by your Extended Memory modules and the configured size of EBTKS provided Extended Memory.

StartBank for HP86A/B, HP87/HP87XM

1. Unplug EBTKS from your system and plug in the extended memory modules you plan to use with EBTKS (64 kB or 128 kB). Note: the 16K RAM module from HP is not compatible with HP86/87 computers.
2. Start your HP86/87 computer and type the **LIST** command to see the amount of memory available. The value should be close to a multiple of 32 kB. For example, on an HP87 that comes with 32 kB installed, the displayed number is 28467 (on my HP87). So this is close to 32768. If I had a 128 kB module plugged in, the number would be close to 159539.
3. To calculate the closest multiple of 32 kB, divide the number provided by the **LIST** command by 32768. For example: 159539 / 32768 results in 4.8687 , so the closest multiple is 5 time 32768. What we care about is the 5 that we just calculated. You can use your HP86/87 computer to do these calculations.
4. Take the result from the prior step, and add 1. In our example we get 6

(Cont'd)

5. The result of step 4 is the value to be used for **StartBank** parameter. Update the CONFIG.TXT file on the SD Card with this value, and plug the SD Card into the socket on the Teensy processor module.
6. EBTKS can now be plugged into your HP86/87 with the Extended memory modules.
7. If you unplug any Extended memory modules, you will have to repeat this whole procedure. Well, actually, if you unplug a 128 kB module, just subtract 4 from the previous calculated StartBank. If you unplug a 64 kB module, subtract 2 from the previous calculated StartBank.
8. Confirm that you have the settings correct by doing a LIST command. The reported available memory should be greater by 32768 times NumBanks.

```
"EMC": {
  "enable": true,
  "NumBanks": 8,
  "StartBank": 2
}
```

Here is an example of CONFIG.TXT

Top

If your browser window is narrow, the text below may be clipped on the right side. If this happens, there is a scroll bar at the bottom of the page that you can adjust to see all the text

```
{
  "machineName": "HP85A",
  "CRTVerbose": true,
  "ram16k": true,
  "screenEmu": true,
  "CRTRemote": true,
  "tape": {
    "enable": true,
    "filepath": "/tapes/tape1.tap"
  },
  "optionRoms": {
    "directory": "/roms85/",
    "roms": [
      {
        "description": "Service ROM 340 AUXROM Aware",
        "filename": "rom340aux",
        "enable": false
      },
      {
        "description": "Assembler ROM",
        "filename": "rom050",
        "enable": false
      },
      {
        "Note": "Do not enable on HP85B as this ROM is built in on mainboard",
```

(Cont'd)

```
          "description": "I/O ROM",
          "filename": "rom300B",
          "enable": false
        },
        {
          "Note": "For original 85A floppies, disable rom317, rom320B, rom321B",
          "description": "Mass Storage",
          "filename": "rom320",
          "enable": false
        },
        {
          "Note 85A": "EBTKS requires this for emulated 5, 10 MB hard disk. Must also enable companion
         "Note 85B": "Do not enable on HP85B as this ROM is built-in on mainboard",
             "description": "85B Mass Storage",
             "filename": "rom320B",
             "enable": true
           },
           {
             "Note 85A": "EBTKS requires this companion to rom320B. Can be used on 85A",
             "Note 85B": "Do not enable on HP85B as this ROM is built-in on mainboard",
             "description": "EDisk",
             "filename": "rom321B",
             "enable": true
           },
           {
             "Note": "For SS/80 disk, with real HPIB and real SS/80 disk. Use with rom320B,r⍰⍰⍰⍰⍰⍰⍰
             "description": "Extended Mass Storage",
             "filename": "rom317",
             "enable": false
           },
           {
             "description": "Advanced Programming",
             "filename": "rom350",
             "enable": true
           },
           {
             "description": "Printer/Plotter",
             "filename": "rom360",
             "enable": true
           },
           {
             "description": "AUXROM Primary 2021_06_25",
             "filename": "rom361",
             "enable": true
           },
           {
             "description": "AUXROM Secondary 1 2021_06_25",
             "filename": "rom362",
             "enable": true
           },
           {
             "description": "AUXROM Secondary 2 2021_06_25",
             "filename": "rom363",
             "enable": true
           },
           {
             "description": "AUXROM Secondary 3 2021_06_25",
             "filename": "rom364",
```

(Cont'd)

```
          "enable": true
        }
      ]
    },
    "hpib": {
      "select": 3,
      "devices": [
        {
          "Comment": "All blocks must have different device numbers",
          "type": 0,
          "device": 0,
          "enable": true,
          "drives": [
            {
              "Comment": "msus$ 300",
              "unit": 0,
              "filepath": "/disks/EBTKS_1.0_85.dsk",
              "writeProtect": false,
              "enable": true
            },
            {
              "Comment": "msus$ 301",
              "unit": 1,
              "filepath": "/disks/85Games1.dsk",
              "writeProtect": false,
              "enable": true
            },
            {
              "Comment": "msus$ 302",
              "unit": 2,
              "filepath": "/disks/Floppy_scr.dsk",
              "writeProtect": false,
              "enable": true
            }
          ]
        },
        {
          "type": 0,
          "device": 2,
          "enable": true,
          "drives": [
            {
              "Comment": "msus$ 320",
              "unit": 0,
              "filepath": "/disks/85Games2.dsk",
              "writeProtect": false,
              "enable": true
            },
            {
              "Comment": "msus$ 321",
              "unit": 1,
              "filepath": "/disks/85GamesMisc.dsk",
              "writeProtect": false,
              "enable": true
            }
          ]
        },
        {
```

(Cont'd)

```
        "Comment": "All blocks must have the same select and different device",
        "type": 4,
        "device": 3,
        "enable": true,
        "drives": [
          {
            "Comment": "msus$ 330",
            "unit": 0,
            "filepath": "/disks/5MB_scr.dsk",
            "writeProtect": false,
            "enable": true
          },
          {
            "Comment": "msus$ 331",
            "unit": 1,
            "filepath": "",
            "writeProtect": false,
            "enable": false
          }
        ]
      },
      {
        "Comment": "Device 10 on HPIB select code, i.e. 310",
        "printer": "",
        "device": 10,
        "enable": true,
        "filepath": "/printers/printfile.txt"
      }
    ]
  },
  "AutoStart": {
    "enable": false,
    "enableTapeAutostart": false,
    "Note": "Max command is 256 characters",
    "command": "\\237\\237LOAD \\042LEDTEST2\\042\\232RUN\\232",
    "batch": ""
  },
  "EMC": {
    "Comment": "Startbank add 2 for each 82908A (64 kB), add 4 for each 82909A (128 kB),
    "enable": false,
    "NumBanks": 2,
    "StartBank": 3
  }
}
```

Disk, Tape and SD Card Storage

EBTKS provides emulated Disk and Tape storage.

Emulated means that there is no actual physical Disk or Disk drive, and no physical Tape or Tape drive. Instead, EBTKS uses a combination of software and hardware on the EBTKS board to create a storage system that from the point of view of a person writing programs on a Series80 computer, and also from the point of view of the built- in software of the Series80 computer, standard commands that access Tapes and Disks operate as if there was one or more Disk Drives and/or a Tape Drive. Neither the standard built-in software or user program can tell that real disk and tape is not connected to the Series80 computer. Configuring the emulated Disk and emulated Tape is done in the CONFIG.TXT file that is located in the root directory of the SD Card that is plugged into EBTKS. The configuration can include disabling the facility, if for example you have a functioning Tape Drive on an HP85A/B.

EBTKS can also co-exist with real Disk Drives, connected via an HPIB module. The only requirement is that the HPIB module select code (default is 7) does not match the select code used by EBTKS (default is 3). Thus, using the standard EBTKS configuration together with a real floppy Disk Drive, the following CAT commands can access both.

Command	What happens
CAT	Display the catalog of “:D300”, an emulated drive on EBTKS
CAT “:D300”	Display the catalog of “:D300”, an emulated drive on EBTKS
CAT “:D700”	Display the catalog of the first real floppy Disk Drive
CAT “:T”	Display the catalog of the emulated Tape Drive on EBTKS

EBTKS can emulate multiple disk drives concurrently, and at most, one Tape Drive.

SD Card Storage using FAT32

Separate from all of the above emulated Disk and Tape storage, EBTKS also provides its own native hierarchical storage on the SD Card, implemented as a FAT32 file system. Unlike the restrictive filename rules that the Series80 computers typically use, EBTKS FAT32 file names can be any length (over 60 would be silly), and this is also true of subdirectory names. There is no practical limit to the depth of subdirectories. The SD Card provides 16 Gigabytes of storage. The FTA32 file is accessed with new keywords provided by the AUXROMS that are part of EBTKS. Follow links on next page to learn more:

(Cont'd)

SDATTR	Returns attribute status for a file or directory - 60
SDCARDMOUNT	Mount/Unmount the SD Card - 58
SDCAT	Display a catalog of the current SD Card directory - 67
SDCD	Change the current SD Card directory - 68
SDCHAIN	Like the normal CHAIN command, but for the SD Cadr - 83
SDCLOSE	Close an SD Card file 63
SDCOPY	Copy an SD Card file to another SD Card file - 62
SDCUR$	Returns the current SD Card directory - 69
SDDEL	Delete an SD Card file - 61
SDEOF	Test an open SD Card file for End Of File - 65
SDEOL	Set the End Of Line character sequence - 78
SDEOL$	Return the End Of Line character sequence - 79
SDEXISTS	Test an SD Card file for existence - 65
SDFFIRST	Initializes reading a directory - 67
SDFLUSH	Flush any un-written data to an open SD Card file - 65
SDFNEXT	Iterative reading of directory entries - 68
SDHOME$	Returns the path for the root directory - 69
SDLOAD	Load a program from the SD Card - 67
SDLOADBIN	Load a binary program from the SD Card - 84
SDMKDIR	Make a new directory on the SD Card - 69
SDMORE	Display a text file, with pagination - 62
SDOPEN	Open an SD Card file - 62
SDPATH$	Return a piece of a path - 83
SDRDLINE	Read from an SD Card file, End Of Line aware - 64
SDREAD	Read from an SD Card file - 63
SDREN	Rename (and/or move) an SD Card file - 61
SDRMDIR	Delete a directory on the SD Card - 69
SDSAVE	Save a BASIC program to the SD Card as ASCII text - 70
SDSEEK	Position the read/write pointer for an open SD Card file - 66
SDSIZE	Returns the size of an SD Card file - 61
SDSLASH	Set the path separation character - 78
SDSLASH$	Report the path separation character - 78
SDSTORE	Store a BASIC program as a LIF file on the SD Card - 66
SDSTOREBIN	Store a Binary program as a LIF file on the SD Card - 84
SDWRITE	Write to an SD Card file - 64

Disk Drive emulation

EBTKS currently emulates 1 HP 82937A HP-IB interface. There can be a number of disk devices each with a maximum of 4 disk drives using the AMIGO protocol.

Each emulated disk drive uses a disk image file stored on the SD card.

There are no changes required on the HP85 side software wise - for all intents and purposes, the HP85 thinks it has a ‘real’ HP-IB card and peripherals.

The select code can be set via configuration in the CONFIG.TXT file that is loaded at boot time.

The emulation is fairly simplistic and only implements the bare minimum to ‘work’. Currently status reporting is not fully implemented. The disk image file format is identical to the one used by Everett Kaser’s emulator - basically a sector by sector image stored in a file.

No benchmarks have been done to compare the performance relative to a ‘real’ disk drive - I would expect it to be significantly faster - no mechanical movement is required, the burst transfers are not throttled and there is no ‘real’ HP-IB to speak of. Many operations appear instantaneous.

There is plenty of scope for improvement - the first would be to complete the AMIGO emulation to give proper status reporting. Then SS/80 might be the next step.

Currently we’ve been using the mini-floppy (82901/9121) as most of the available disk images are for this format, and 5 MB hard disks (emulated winchester disk drives). EBTKS also supports (but not yet tested) emulation of 8” floppy disks.

For those that want to delve into the code:

HPDisk.h has the class that implements a single disk drive. The mapping of tracks/sectors for the disk drive types is done here.

HPIBDisk.h has the class that implements a disk drive device. It has a collection of HPDisk objects (up to 4 - HP s/w limitation) and forms the virtual HPIB device. The AMIGO protocol is implemented here.

EBTKS_1MB5.cpp is where the virtual hardware HPIB interface is implemented. The 1MB5 translator chip and the Intel i8049 processor (the HP custom chip and microprocessor used in the real interface) are emulated - but not completely. Only the bare minimum is done - the goal was a functional implementation rather than a cycle and function perfect emulation. Up to 31 instances of the HPIBDisk class are supported.

The basic hierarchy:

```
                                      EBTKS_1MB5
                                          |
                  +-----------------------+--------------------+
                  |                                            |
              HPIBDisk0                                    HPIBDisk1
                  |                                            |
   +---------+----+----+---------+              +---------+----+----+---------+
   |         |         |         |              |         |         |         |
HPDisk0   HPDisk1   HPDisk2   HPDisk3        HPDisk0   HPDisk1   HPDisk2   HPDisk3
```

The code has functions that emulate the 1MB5 hardware registers that are called via the bus Interrupt Service Routine (ISR). The emulation is split between the real-time functions (responding to ISR events) and the non-realtime processing (which represents the bulk of the work) and is processed in the main background processing loop. Due to this, the interface is rather complex due to the operation of the 1MB5 - there's interrupts, acknowledges, burst transfers and timing sensitivities. Tread carefully if you want to modify this! As well, the code cannot block as there are other tasks in the background loop that may need attention.

If you're feeling adventurous and like a challenge - there's always the possibility of writing a new ROM to implement the filesystem directly on the SD card and avoid the HPIB/AMIGO stuff.

The emulation does not create blank disks. Instead we copy a reference file that has the required initialization to achieve this, using the MOUNT command.

Tape drive emulation

Like the disk drives, the tape is a functional emulation. i.e. it doesn't attempt to precisely mimic a real tape. It does enough to keep the HP85 code happy to think it has a tape unit. The tape emulation file format is identical to Everett Kaser's HP emulator. This mimics the real tape as an array of bytes with flags to emulate the tape drive motor tachometer slots. Due to this the file is 800k or so bytes for a 120 KB tape. On the other hand, we have 16 GB of storage on the SD Card.

Due to the emulation method, the emulation is faster than the real one, but we still have to traverse the virtual tape so it is not much faster. Like the real tape drive, the CRT is blanked while the virtual motor is moving the virtual tape past the virtual read/write head. We do not implement a virtual failed capstan rubber roller, or a virtual failed tape cartridge drive band.

The emulation uses a windowing method where a block of tape data is loaded into and out of memory from the SD card file. As the tape is traversed, the required block is loaded. This is done to preserve the amount of ram required for the tape emulation.

The emulation does not create blank tapes. Instead we copy a reference file that has the required initialization to achieve this, using the MOUNT command.

(Cont'd)

> No write protection is implemented. The emulation hasn’t been exhaustively tested - especially with the corner cases like a full tape and extensive file operations.
>
> We don’t expect much interest in the tape - once you have disk functionality, the tape becomes a historical curiosity. Which is kind of sad, given that tape emulation was the original purpose of EBTKS.
>
> The code for the tape emulation is in the file EBTKS_Tape_Drive.cpp
>
> The CRT blanking during Tape operations is built into the HP85 system ROM code. HP did this to reduce the load on the power supply.

16 KB RAM for the HP-85A

EBTKS can be configured in the CONFIG.TXT file to provide 16 KB of RAM in the upper part of a HP85A computer (and probably also HP83).

This capability should not be enabled in HP85B, or HP86/87 as they already implement this memory area.

AUXROM Keywords

AUXERRN	AUXROM Error number when Error 109 is reported - 71
AUXREV	Returns the AUX ROMs revision number - 79
BOOT	Restarts EBTKS and the Series80 Computer - 81
CRTCOLS	Returns # of columns on CRT display - 77
CRTCURSCOL	Returns cursor column number - 77
CRTCURSOR	Sets cursor to REPLACE/INSERT - 77
CRTCURSROW	Returns absolute cursor row number - 78
CRTGETTOP	Returns the row# of top of the CRT display - 76
CRTON	Blanks/Unblanks CRT - 77
CRTREAD$	Reads from CRT - 76
CRTROWS	Returns # of rows on CRT display - 77
CRTSETTOP	Sets CRT start row - 76
CRTWRITE	Writes to CRT - 76
DATETIME	Read the Real Time Clock - 71
EBTKSREV$	Returns a string with EBTKS firmware build date and time - 79
HELP	Built-in HELP (in development) - 75
KBDBUFFER	Takes over keyboard - 75
KBDISKEY	Test if keys in buffer - 75
KBDKEY	Returns the next keycode from the keyboard buffer - 75
LISTROMS	List all ROMs or just active ROMs - 73
MEDIA$	Returns the name of a LIF image - 60
MOUNT	Mount a LIF image on a msus$ - 58
PEEK	Read a byte from memory or I/O space - 74
POKE	Write a byte to memory or I/O space - 74
RPEEK	Read a byte from a bank switched ROM - 74
RPOKE	Write a byte to a bank switched ROM (dangerous) - 74
RSECTOR	Reads a raw 256-byte sector - 81
SDATTR	Returns attribute status for a file or directory - 60
SDBATCH	Start reading keyboard input from a batch file - 84
SDCAT	Display a catalog of the current SD Card directory - 67
SDCD	Change the current SD Card directory - 68
SDCHAIN	Like the normal CHAIN command, but for the SD Card - 83
SDCLOSE	Close an SD Card file - 63
SDCOPY	Copy an SD Card file to another SD Card file - 62
SDCUR$	Returns the current SD Card directory - 69
SDDEL	Delete an SD Card file - 61
SDEOF	Test an open SD Card file for End Of File - 65
SDEOL	Set the End Of Line character sequence - 78
SDEOL$	Return the End Of Line character sequence - 79

(Cont'd)

SDEXISTS	Test an SD Card file for existence - 65
SDEXPORTLIF	Exports a file from a LIF disk to an SD Card file - 81
SDFFIRST	Initializes reading a directory - 67
SDFLUSH	Flush any un-written data to an open SD Card file - 65
SDFNEXT	Iterative reading of directory entries - 68
SDGET	Read a BASIC source file from the SD Card - 70
SDHOME$	Returns the path for the root directory - 69
SDIMPORTLIF	Imports a LIF file from the SD Card and write it to a LIF disk - 82
SDLOAD	Load a program from the SD Card - 67
SDLOADBIN	Load a binary program from the SD Card - 84
SDMKDIR	Make a new directory on the SD Card - 69
SDMORE	Display a text file, with pagination - 62
SDOPEN	Open an SD Card file - 62
SDPATH$	Return a piece of a path - 83
SDRDLINE	Read from an SD Card file, End Of Line aware - 64
SDREAD	Read from an SD Card file - 63
SDREN	Rename (and/or move) an SD Card file - 61
SDRMDIR	Delete a directory on the SD Card - 69
SDSAVE	Save a BASIC program to the SD Card as ASCII text - 70
SDSEEK	Position the read/write pointer for an open SD Card file - 66
SDSIZE	Returns the size of an SD Card file - 61
SDSLASH	Set the path separation character - 78
SDSLASH$	Report the path separation character - 78
SDSTORE	Store a BASIC program as a LIF file on the SD Card 66
SDSTOREBIN	Store a Binary program as a LIF file on the SD Card - 84
SDWRITE	Write to an SD Card file - 64
SETLED	Set the color and intensity of the two RGB LEDs - 80
SPRINTF	Complete control of formatted output, using C sprintf() - 72
UNMOUNT	Unmount the LIF image from a msus$ - 59
WSECTOR	Writes a raw 256-byte sector - 81
AUXBUF$	Low Level AUXROM Interface for developers - 87
AUXCMD	Low Level AUXROM Interface for developers - 86
AUXOPT$	Low Level AUXROM Interface for developers - 86

AUXROM

AUXROM is a set of 8KB ROM images that are the companion firmware for the HP Series 80 computers when the EBTKS board is installed. The primary author of AUXROM is Everett Kaser, and the companion software on EBTKS that AUXROM communicates with was authored by Philip. This page documents the functions that AUXROM implements and also covers the internal architecture of how AUXROMs and EBTKS communicate

(Cont'd)

The AUXROM(s) like all other optional ROMS on the HP-85 execute in an address space window from 060000 to 077777 (octal). This is 8K bytes. At any time only one option ROM can be active, and the ROM is selected by an I/O register named RSELEC. In almost every other respect, the AUXROM is different than any ROM for the HP-85. Some of these differences will be seen by the user, and some just make what happens behind the curtain easier to implement, or allow much faster operation. See Memory Map for more details

Compatibility with Everett’s Series 80 Emulator

Many of these features are available in Everett Kaser’s Series 80 Emulator, as well as in a real HP-85 with an EBTKS. Some functionality may be slightly different between the two environments, due to differences in the environments.

BEWARE: With great power comes great responsibility. You could very easily (were you a person liable to moments of criminal carelessness) delete entire LIF volumes, your configuration file, or your entire hard disk (if you’re running this in the Emulator). Be wary.

AUXROM Function Overview

AUXROM is identified as ROM 361 (octal), but it may also claim adjacent ROM numbers up to 376 (octal), but these additional IDs do not get detected when the HP-85 starts up, and they do not add any new function names or keywords.

The AUXROM has many functions for accessing and manipulating files stored on the 16 GB SD card’s FAT32 file system. Most of these new Keywords can be included in your programs. They provide the ability to directly use the FAT32 file system, including reading, writing, creating, and deleting both files and directories.

File names can be as long as you want (within reason), and you can have nested directories, with long sub-directory names.

The FAT32 file access functions are a reasonable subset of the Posix file manipulation functions.

For most of the SDxxxx keywords that involve a filename or path, the system (EBTKS or Emulator) keeps track of the “current path,” much as MS-DOS or a Windows command line or a unix shell keeps track of a current working directory. The filenames or paths specified in these SDxxxx keywords (functions and statements) may be ABSOLUTE paths (ones starting with a ‘/’ or \’\\’), or they may be RELATIVE (NOT starting with a slash). RELATIVE paths will automatically be appended, internally, to the “current path” to make them an ABSOLUTE path. By default, the path separator is ‘/’

Unquoted file paths and file names

In some SDxxxx statements, the filename can be specified either quoted or unquoted. You can’t use unquoted if the filename contains a SPACE, @, $, COMMA, or EXCLAMATION POINT character. If it contains one of those, then you’ll have to ” ” the filename to avoid confusing the parser. The SDxxxx statements that will accept either a quoted or unquoted string are:

- SDMKDIR
- SDRMDIR

(Cont'd)

- SDDEL
- SDCD
- UNMOUNT
- SDSAVE
- SDGET
- SDMORE
- SDLOAD
- SDSTORE

These statements all have one string argument, with no options for other arguments that could confuse the parser. Other SDxxxx statements that have more than a single string argument MUST use quoted strings.

AUXROM Error messages

In all of the following documentation, error status is returned in the following ways.

Standard error messages are from the system ROMs and have values from 1 to 92 and are listed in Apendix E of the HP85A Owner's manual and Programming Guide (page 307). The HP85B manual has the same information (page 381), and the HP86/87 manual has the same information (page 343). EBTKS uses these error numbers where appropriate, and the error numbers are returned in ERRN.

Standard EBTKS error messages are from 108 (a warning) through to 128 (currently). These error numbers are fairly general and are shared across many EBTKS Keywords. The error numbers are returned in ERRN.

Custom EBTKS error messages use the Standard EBTKS error messages 109 as a starting point, then add an error message that is specific to the keyword. These error messages display as follows:

Error 109 : Custom text message

For these error messages, as well as ERRN being set to 109, an additional variable AUXERRN is set to a number that is specific to the custom error message. These start at 300. For each keyword on the rest of this page, when an error messages is shown with custom text and an AUXERRN value, it is reported as per the above example, with ERRN set to 109 and AUXERRN set to the listed value.

Loading and Unloading emulated LIF disk and tape images

Throughout this page, the references to msus$ is the same as the traditional Series 80 usage. See the HP Series 80 documentation for details of legal msus$ strings.

For MOUNT and UNMOUNT, the "msus$" may refer to EITHER an emulated disk drive (:Dxxx) OR the emulated TAPE drive (:T). The MEDIA for an emulated disk drive is an SD file that emulates the contents of a disk, whereas the MEDIA for the emulated tape drive is an SD file that emulates the contents of a tape. These are NOT interchangeable: a MEDIA created for one device (disk or tape) cannot be loaded into the other emulated device. If the MEDIA gets created, it will be the proper length (containing the proper amount of physical

(Cont'd)

‘records’), and it will be ‘initialized’. There is no need to INITIALIZE a new disk image or ERASETAPE for a new tape image.

NOTE: With Everett Kaser’s HP8X Emulator, all disk images must exist in one of
the DISKS0 to DISKS9 sub-folders. For the emulator, Disk images need to be INITIALIZE-ed and Tape images need to be ERASETAPE-ed.

There is also a special mode to Mount/Unmount the SD Card.

MOUNT msus$, filePath$ [, modeFlag]

Top

Associate the specified LIF image file with msus$

- msus$ specifies the emulated disk/tape drive that will be associated with the LIF image
- filePath$ is the name (maybe with path) of the emulated MEDIA (LIF image). Typically disk LIF images are in the /disks/ directory, and tape images are in the /tapes/ directory.
- modeFlag controls what happens if emulated MEDIA exists or not
  - 0 = error if doesn’t exist, else MOUNT (this is the default if not specified)
  - 1 = if exists, MOUNT, else create blank and MOUNT
  - 2 = if exists, error, else create blank and MOUNT
- There is a special case of the msus$ parameter being “SDCARD” to mount the SDCard, see below.

Possible Errors

- “Can’t resolve path” , AUXERRN will be 330
- “Invalid MOUNT mode” , AUXERRN will be 410
- “MOUNT file does not exist” , AUXERRN will be 411
- “MOUNT MSU$ error” , AUXERRN will be 412
- “MOUNT Filename must end in .tap” , AUXERRN will be 413
- “MOUNT HPIB Select must match” , AUXERRN will be 414
- “MOUNT failed” , AUXERRN will be 415
- “MOUNT Filename must end in .dsk” , AUXERRN will be 416
- “Couldn’t open Ref Disk” , AUXERRN will be 417
- “Couldn’t open New Disk” , AUXERRN will be 418
- “Couldn’t init New Disk” , AUXERRN will be 419
- “While MOUNTing an SD card, failed”, AUXERRN will be 406
- “Mounting virtual drives failed”, AUXERRN will be 407
- “Couldn’t read Ref Disk” , AUXERRN will be 408
- “MOUNT File already exists” , AUXERRN will be 409
- “Couldn’t open Ref Tape” , AUXERRN will be 426
- “Couldn’t open New Tape” , AUXERRN will be 427
- “Couldn’t init New Tape” , AUXERRN will be 428
- “Device code not supported”, AUXERRN will be 512

Examples

- MOUNT “:D300”, “/disks/85AssemblerROMdisc.dsk”

(Cont'd)

Uses 0 as the default modeFlag.

- MOUNT “:D300”, “/disks/NewDisk.dsk”, 2

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

- MOUNT “:T”, “/tapes/DataTape.tap”, 1

  If “DataTape” exists, MOUNT it, otherwise create a blank SD file called “DataTape” and mount it.

UNMOUNT msus$

Top

- msus$ specifies the emulated disk or tape drive from which to remove the current ‘emulated’ MEDIA file.
- There is a special case of the msus$ parameter being “SDCARD” to unmount the SDCard, see below.

This Keyword supports unquoted names (but still needs the :D or equivalent)

Possible Errors

- “UNMOUNT MSU$ error” , AUXERRN will be 490
- “UNMOUNT Disk error” , AUXERRN will be 491

Examples

- 10 UNMOUNT “:D300”
- 20 UNMOUNT “:T”

Mounting and Unmounting the SD Card

Top

MOUNT and UNMOUNT also support an msus$ of “SDCard” (not case sensitive) that allows removing and installing the SD Card without having to turn the power to the HP85 off and back on. For the MOUNT keyword, it requires a second parameter, which for this usage, can be any string or string variable, since the value is ignored.

- UNMOUNT “sdcard”

will close all the LIF files for each of the current emulated disk and tape drives. After this keyword is run, it is safe to unplug the SD Card.

- MOUNT “SDCard”, “A”

This keyword should be run after the SD Card is plugged back into the EBTKS. The CONFIG.TXT file is read, and unlike system boot where all the sections of the file are used (setting options, loading ROMs, setting up virtual disks, etc…), re-mounting the SD Card and running the above line, will only process the disk and tape specifications, re-associating the relevant LIF files with their respective msus$. The second parameter is needed to meet the requirements of the Mount keyword, but in this case it isn’t used. Any non-empty string will do.

MEDIA$(msus$)

Top

Report the filename of the LIF image associated with the msus$

- msus$ specifies the emulated disk or tape drive for which the currently ‘emulated’ MEDIA file’s name should be returned.

Return Value

- Returns a string that is the full path to the LIF image

Possible Errors

- “MEDIA$ MSU$ error” , AUXERRN will be 510
- “MEDIA$ HPIB Select must match” , AUXERRN will be 511
- “Device code not supported” , AUXERRN will be 512

Examples

- A$ = MEDIA$(“:T”)
- A$ = MEDIA$(“:D301”)

SD Card File Manipulation

SDATTR(filePathName$)

Top

Returns a value that encodes the Attributes of a file or a directory. There are only two types of attributes:

- Whether the object is a file or a directory, encoded in bit 0
- Whether the object is normal access, or it is read-only, encoded in bit 1

The two independent attributes are combined into a single returned integer in the range 0 to 3

Return Value

Bit 1	Bit 0	Return Value	Meaning
0	0	0	filePathName$ is a normal read/write access **file**
0	1	1	filePathName$ is a normal **directory**
1	0	2	filePathName$ is a **file** with read-only access
1	1	3	filePathName$ is a **directory** with read-only access

Possible Errors

- “SD Error” ERRN will be 113. Bad file/pathname or includes wild cards or file/path name not found

Examples

- Example: A = SDATTR(“ZINK”)

SDSIZE(filePathName$)

Top

Return Value

- Returns the size of the specified SD file.

Possible Errors

- “SD Error” ERRN will be 113. Bad file/pathname or includes wild cards or file/path name not found

SDDEL   fileSpec$

Top

Deletes the specified SD file (does not work on sub-directories, yet).
Wild cards (* and ?) may be used in the filename part of fileSpec$, but not in the path part.

This Keyword supports unquoted names

Possible Errors

- “Can’t resolve path” , AUXERRN will be 330
- “SDDEL no file specified” , AUXERRN will be 370
- “Couldn’t delete file” , AUXERRN will be 371
- “SDDEL no wildcards in path” , AUXERRN will be 372
- “SDDEL problem with path” , AUXERRN will be 373

SDREN   oldName$, newName$

Top

Renames the specified file or directory to the new name. Can be used to move files between directories

Possible Errors

- “Can’t resolve Old path” , AUXERRN will be 450
- “Can’t resolve New path” , AUXERRN will be 451
- “SDREN rename failed” , AUXERRN will be 452
- “SDREN Mystery bug” , AUXERRN will be 453
- “SDREN Old file doesn’t exist” , AUXERRN will be 454

SDCOPY   SourceName$, DestinationName$ [, Overwrite]

Copies the specified Source file to the specified Destination file. If the Overwrite option is not given, or it is 0, then overwriting an existing file is an error. If the Overwrite option is provided and is 1, then an existing file will be overwritten/replaced.
Does not support wild cards, yet.
Does not support copying a complete directory, yet.
SourceName$ and DestinationName$ must be string variables or quoted strings.

Possible Errors

- “Can’t resolve Source path” , AUXERRN will be 520
- “Can’t resolve Destination path” , AUXERRN will be 521
- “SDCOPY bug in code” , AUXERRN will be 522
- “Source file doesn’t exist” , AUXERRN will be 523
- “Couldn’t open Source File” , AUXERRN will be 524
- “Couldn’t open Destination File” , AUXERRN will be 525
- “File copy failed” , AUXERRN will be 526
- “SDCOPY File already Exists” , AUXERRN will be 527

SDMORE   SourceName$ [, Paginate]

Top

Displays (types) the SD file to the CRT, optionally ‘paginating’ it (ala the unix MORE command) if ‘paging’ is 1. If ‘paging’ is not specified, then the default is 0 (no paging). This dumps the file to the CRT as fast as it can, and it is NOT redirectable by using the CRT IS statement, it will still go to the CRT. When paging and paused at a page boundary, the RUN key will dump the rest of the file with no further paging/pausing, while -LINE, BACKSPACE, -CHAR, and PAUSE will quit the SDMORE immediately.

This Keyword supports unquoted names

Possible Errors

- “Open failed Mode 0” , AUXERRN will be 422
- “Invalid Mode” , ERRN will be 114 , Paginate should be 0 or 1

SD Card File Access

SDOPEN   filePathName$, mode#, file#

Top

Opens a file on the SD card for read/write access. Three different modes support opening existing files for reading-only, or for reading & writing, or for creating or truncating the file. The file is always opened in ‘binary’ mode, no ‘text’ mode is supported, except via the SDRDLINE statement.

(Cont'd)

- filePathName$ may be an absolute path to the file or a path relative to the current folder (returned via SDCUR$ and set via SDCD).
- mode#
  - 0 = Error if doesn't exist, else open for READ-ONLY set to START of file.
  - 1 = Open if exists, create if doesn't, set position to END of file.
  - 2 = Truncate if exists, create if doesn't, set position to START of file.
- file#:

  A number from 1 to 10. SD file access supports having up to 10 files open at the same time, and each one gets assigned to a file# from 1 to 10. These numbers are different and not related to the Basic ASSIGN buffer numbers, although they serve the same general purpose, just for the SD file system rather than for the Series 80 tape and disk file system.

Possible Errors

- "File is already open" , AUXERRN will be 420
- "Parsing problems with path" , AUXERRN will be 421
- "Open failed Mode 0" , AUXERRN will be 422
- "Open failed Mode 1" , AUXERRN will be 423
- "Open failed Mode 2" , AUXERRN will be 424
- "Open failedIllegal Mode" , AUXERRN will be 425
- "Couldn't open Ref Tape" , AUXERRN will be 426
- "Couldn't open New Tape" , AUXERRN will be 427
- "Couldn't init New Tape" , AUXERRN will be 428

SDCLOSE   file#

Top

Closes a previously opened file.

- file# is a file number from 1 to 10 that was used to open the file.
- file# is 0 to close all open files

Possible Errors

- SD ERROR (213D) if file# was not open ##### needs to be fixed once Auxroms fixed #####

Examples

- 10 SDCLOSE 0
- 20 B = 4 @ SDCLOSE B

SDREAD   dst$Var, bytesReadVar, maxBytes, file#

Top

Reads some number of bytes from the already opened SD file, placing the bytes into the destination string variable and the number of bytes read into the bytesReadVar. The number of bytes read will be either maxBytes

(Cont'd)

or the remaining number of bytes in the file, whichever is smaller.

```
dst$Var        Where the data that is read is stored
bytesReadVar   Variable that is set to the actual number of bytes read
maxBytes       Requested number of bytes to read
file#          File Number
```

Possible Errors

- “SDREAD File not open” , AUXERRN will be 440

Example

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

SDRDLINE   dst$Var, bytesReadVar, maxBytes, file#

Top

Reads bytes from the file into dst$Var until one of the following occurs:

- maxBytes have been read (Note: can’t be greater than 1500)
- the end-of-file is reached
- an EOL-sequence (CR/LF or LF or CR) is seen

Sets bytesReadVar to the number of bytes stored into dst$Var. The EOL-sequence is NOT stored in dst$Var. Returns -1 if at end of file.

Possible Errors

- SD ERROR (213D) if file# not open, or general SEEK or READ error.

Example

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

SDWRITE   src$, file#

Top

Writes a string to a previously opened SD file. This is a binary mode file operation, no EOL-sequence is automatically written. If you are writing text and wish an EOL-sequence, use SPRINTF and SDEOL$ to format the output string with an EOL-sequence on the end before writing the formatted string to the file.

Possible Errors

- “SDWRITE File not open for write” , AUXERRN will be 480

(Cont'd)

Example

```
DIM F$[600]
     N "ZINK",2,5
     TF F$, "Hello, world!%s", SDEOL$
SDWRITE F$, 5
```

SDEOF(file#)

Top

A numeric function that tells you how many bytes from the current “file position pointer” to the “end of the file”

Return Value

- Returns 0 if the current file# position is at the end-of-the-file
- Returns a positive number if there are bytes from the current file position to the end of the file.

Possible Errors

- SD ERROR (213D) if file# is not open or a general SEEK error occurs.

Example

```
(This needs to be checked)
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
```

SDEXISTS(filePathName$)

Top

Return Value

- Returns 0 if the specified file or sub-dir does not exist, otherwise returns 1.

Possible Errors

- none

SDFLUSH   file#

Forces any pending (buffered) writes to be flushed to storage. This has no effect when running on the Series 80 Emulator, but when running on the EBTKS, it causes buffers to be written to the SD card. If the file# is 0, then **all** open files are flushed.

Possible Errors   •none

SDSEEK(mode#, offset#, file#)

Top

Moves the “current position” pointer in a file to a specified location. The new location can be specified in one of three relative offsets as specified by the mode#:

Return Value

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

Return Value

- New file position

Possible Errors

- “Seek on file that isn’t open” , AUXERRN will be 470
- “Trying to seek before beginning” , AUXERRN will be 471
- “Trying to seek past EOF” , AUXERRN will be 472
- “SDSEEK failed somehow” , AUXERRN will be 473

SDSTORE nameSD$

Top

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”.

This Keyword supports unquoted names

Possible Errors

- FILE/PATH (217D) error in file/path name, or error opening/creating the file
- SD ERROR (213D) if write failed to write everything

Top

SDLOAD   nameSD$

Loads a Basic program into memory from an SD file that was SDSTORE'd.

This Keyword supports unquoted names

-

Possible Errors

FILE/PATH (217D) error in file/path name, or error opening/creating the file

Directory Manipulation

SDCAT   [fileSpec$]

Top

Displays a catalog of the SD file system. If no fileSpec$ is provided, "*" is used. fileSpec$ is a string expression which may or may not include wild card characters ('*' and '?'). 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 because it's too long to display, the last character that is shown will be underlined. If the file is READONLY, the size value will be underlined. If it's a sub-directory, a '/' is shown for the size, and if it's a READ-ONLY sub-directory the '/' will be underlined.

- SDCAT
- SDCAT fileSpec$

Possible Errors

- "Can't resolve path" , AUXERRN will be 330
- "Can't list directory" , AUXERRN will be 331
- "SDCAT no wildcards in path" , AUXERRN will be 333

SDFFIRST   name$, date$, size, attrib, fileSpec$

Top

This keyword initializes an iterative process that delivers catalog lines 1 per call. This command delivers the first catalog line that matches the fileSpec$, and the next command (SDFNEXT) is use to deliver the remainder by repetitive calls. The fileSpec$ is only needed for this keyword. It is remembered for the subsequent SDFNEXT calls. There can only be one active iterative catalog process. If another SDFFIRST occurs before the calls to SDFNEXT have exhausted the current catalog, a new iterative process is started.

- SDFFIRST name$, date$, size, attrib, fileSpec$

Return Values

```
name$    The first catalog entry that matches fileSpec$
         An empty string if no catalog entries matches the fileSpec$
date$    The associated date and time
size     The file size (0 if a directory)
attrib   The file attributes
```

See this for a description of attributes

(Cont'd)

Possible Errors (due to implementation details, the possible error messages are shared with SDCAT)

- “Can’t resolve path” , AUXERRN will be 330
- “Can’t list directory” , AUXERRN will be 331
- “No SDCAT init or past end” , AUXERRN will be 332
- “SDCAT no wildcards in path” , AUXERRN will be 333

SDFNEXT   name$, date$, size, attrib

Top

This keyword must be preceded by a call to SDFFIRST to set up the fileSpec$ (that may include wild cards).

The return values are the same as SDFFIRST. Successive calls to this keyword will return successive catalog entries until all have been returned. After the last entry has been returned, the next call will return a name$ that is a string of length 0. This concludes the iterative retrieval of catalog entries. If you make an additional call to SDFNEXT after it has indicated that there are no more entries (length 0 for name$), then an error is reported.

Possible Errors (due to implementation details, the possible error messages are shared with SDCAT)

- “Can’t resolve path” , AUXERRN will be 330
- “Can’t list directory” , AUXERRN will be 331
- “No SDCAT init or past end” , AUXERRN will be 332
- “SDCAT no wildcards in path” , AUXERRN will be 333

SDCD   pathDir$

Top

Changes the “current SD directory” to pathDir$. pathDir$ may be a relative path (relative to the CURRENT “current directory,” or it may be absolute (starting with a SLASH).

This Keyword supports unquoted directory paths

Possible Errors

- “Can’t resolve path” , AUXERRN will be 330
- “Unable to open directory” , AUXERRN will be 340
- “Target path is not a directory” , AUXERRN will be 341
- “Couldn’t change directory” , AUXERRN will be 342

SDCUR$

Return Value

- Returns a string containing the current full (absolute) SD path.

Possible Errors

- none

SDHOME$

Top

Return Value

- Returns a string containing the original starting full (absolute) SD path, the “home” directory. This NEVER changes during any given “power/run cycle.” This is always “/” on EBTKS, but is a path on the Everett Kaser Series 80 Emulator

Possible Errors

- none

SDMKDIR   folderName$

Top

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. In other words, you can only create ONE LEVEL of sub-directory at a time.

This Keyword supports unquoted names

Possible Errors

- FILE/PATH (217D) if too long or invalid
- SD ERROR (213D) if failed
- “SDMKDIR Dir already exists” , AUXERRN will be 400 (maybe some other error)

SDRMDIR   folderName$

Top

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

This Keyword supports unquoted sub-directories

Possible Errors

- FILE/PATH (217D) if too long or invalid
- SD ERROR (213D) if failed

Export/Import Of Series 80 Files

SDSAVE   filePath$

Top

Saves the currently loaded program to an ASCII SD file (essentially, it lists the program to the file, appending the current SDEOL sequence to the end of each line. This is a non-programmable command.

This Keyword supports unquoted names

Possible Errors

- FILE/PATH (217D) error in file/path name, or error opening/creating the file various system listing errors, primarily ROM MISSING

SDGET   filePath$

Top

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. This is a non-programmable command.

This Keyword supports unquoted names

Possible Errors

- FILE/PATH (217D) error in file/path name, or error opening the file
- BUSY (221D) text buffering is already in use (only one at a time)
- Various system parsing errors (could be almost anything if the file was hand edited)

NOTE 1:

> SDGET does **not** do a SCRATCH, so the SDGET'd lines will be edited into the currently in-memory program. If you want a clean slate, do a SCRATCH first yourself.

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 (unless you can remove enough spaces to shrink the line down to what will fit on three lines of the display). Also, 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. So it goes.

GENERAL FEATURES

DATETIME Year, Month, Day, Seconds

Top

Returns four numeric values from the Real Time Clock (RTC).

- Seconds is the number of seconds since midnight. Range is 0 to 86399
- Day is the day of the month. Range is 1 to 31
- Month is the month of the year. Range is 1 to 12
- Year is the current year. Range is 1970 to 2038

For example

```
DATETIME A,B,C,D
DISP "Year";A
DISP "Month";B
DISP "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
```

Possible Errors

- none

AUXERRN

Top

The AUXROM supports the ability for EBTKS to create custom error messages that are not in the original ROM. These custom error messages always have an error number of 109. To make it easier to distinguish errors (in a program), if you have an ON ERROR statement active, then the handler routine for the ON ERROR can check the error number for 109, and if so, check ERROM for 241 (the decimal# of the AUX ROM), and if so, it can then use AUXERRN to get a unique value that is set by the EBTKS error code, allowing you to easily differentiate between differentiate (programmatically) ‘custom’ error messages.

Possible Errors

- none

Top

SPRINTF dst$Var, format$ [, comma-separated-arg-list]

A reasonable facsimile of the **C** language/library sprintf() function. ‘dst$Var’ is the string into which the formatting output will be stored. ‘format$’ is the string that specifies the formatting, which will use the arguments in the comma-separated-arg-list to sequentially fill formatting parameters in the ‘format$’. The comma-separated-arg-list entries can be numeric or string expressions, although the type **must** match the ones specified in ‘format$’.

NOTE:

> The behavior of this code is **very** dependent upon the library used. The behavior of SPRINTF may very possibly differ between EBTKS and the Series 80 Emulator, due to different sprintf() library routines being linked in for the different environments. Most of the more ‘common’ sprintf() options should behave the same and be supported. However, some of the less common ones may work differently **or** not work at all. Beware.

The format$ string will simply output any normal text included, but whenever it sees a ‘%’ character, that will start the formatting of an argument from the arg-list (unless the ‘%’ is immediately followed by another ‘%’, in which case the two are replaced by a single ‘%’ character in the output and no arg-list items are used). The general format of the % formatting is:

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

NOTE:

> The []’s in the above indicate **optional** things, they are not included in your format$.
>
> [flags] can be any (or none) of the following

```
-       left-align output rather than right-align output (the default)
+       prepends a plus for positive signed-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
#       alternate forms:
           for g and G types, trailing zeros are not removed
           for f, F, e, E, g, G types, the output always contains a decimal point.
           for o, x, X types, 0, 0x, 0X respectively is prepended to non-zero numbers.
```

.

> [width] is a number that specifies the MINIMUM NUMBER of characters to output, used to pad output of smaller numbers; no truncation, though, of numbers too large for the width specified.
>
> [.precision] is a number that specifies a MAXIMUM limit on the output, depending upon the ‘type’.

NOTE:

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

```
'type'       is a single letter indicating the desired formatting of the next item
             from the arg-list
i or d       format the next argument (MUST be numeric) as a signed integer.
u            formats the next argument (MUST be numeric) as an unsigned integer.
f or F       formats the next argument (MUST be numeric) as a REAL in fixed-point
             notation. The only difference is whether VERY large or VERY small numbers
             are output as upper or lowercase INF, INFINITY, or NAN.
e or E       formats the next argument (MUST be numeric) in standard
             "[-]d.ddd e[+/-]ddd" form. The only difference is the case of the 'e' or
             'E' used for the exponent.
g or G       format the next argument (MUST be numeric) in either fixed-point or
```

(Cont'd)

```
             standard-exponential format, whichever is more appropriate for the magnitude.
x or X       formats the next argument (MUST be numeric) as a hexadecimal value.
o            formats the next argument (MUST be numeric) as an octal value.
s            copies the next argument (MUST be string) 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 also include special characters in the output by placing these character
strings in the 'format$':
  \\    outputs a single '\' character
  \r    outputs a CR character
  \n    outputs a LF character
  \t    outputs a TAB character
  \xHH  outputs a character who's value is specified by the two HH hexadecimal digits
  \nnn  outputs a character who's value is specified by the three nnn octal digits
```

For example

```
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$, "\t%s", SDEOL$
```

Possible Errors

- SD ERROR (213D) formatted output length > 1024 characters
- BAD FORMAT (219D) something wrong with format$
- FORMAT/ARG MISMATCH (220D) argument list doesn’t line up with format$

LISTROMS   0 | non-0

Top

If the supplied argument is 0, then a reference listing will be sent to the CRT IS device, showing all known Series 80 ROMs, their ROM numbers (in both octal and decimal), and indicate whether each ROM was available for the HP-85 class of machines and/or for the 87 class of machines.

If the supplied argument is non-0, then the same listing will be shown, but for ONLY the ROMs that are currently present in the system.

When non-zero is used to list the INSTALLED ROMs, only AUX ROM 1 (361) will be listed, as it’s the only one in the systems ROM table. But if AUX ROM 1 is there, you can count on AUX ROMs 2 (362) and 3 (363) being there as well, or you’ll get an error when you try to use any AUX ROM features.

Possible Errors

- none

POKE address, byteVal

Top

Writes the ‘byteVal’ to the ‘address’ in memory. Both values are decimal values. You can use the Assembler ROM’s DEC() function to specify values in octal.

Possible Errors

- none

PEEK(address)

Top

Return Value

- Reads a byte from memory at ‘address’ and returns it as the function value. The ‘address’ is specified in decimal and the returned value is decimal. The Assembler ROM’s DEC() and OCT() functions can be used to convert.

Possible Errors

- none

RPOKE rom#, address, byteVal

Top

Same as POKE, except a ROM number can be specified, which is useful for writing bytes to a bank-swapped ROM, which is normally not terribly useful, since ROMs can’t be written to. However, the AUX ROMs are special in that they have 3 KB of RAM in the middle of the ROMs that is used for the AUX ROM code to communicate with EBTKS (and vice versa). So this function (and RPEEK) can be useful to AUX ROM / EBTKS developers.

Possible Errors

- none

RPEEK(rom#, address)

Top

Return Value

- Reads a byte from the specified bank-selected ROM.

Possible Errors

- none

HELP   [optional quoted or unquoted string]

Top

EBTKS will have help screens that provide various information about the Series 80 computers, their keywords, and whatever people write up. The HELP keyword lets you request the help screen of your choice, and (assuming it's found) EBTKS will save the current CRT contents, display the help, then restore the CRT contents when you press a key.

Possible Errors

- NOT FOUND (224D) if requested help screen is not found

Keyboard Functionality

KBDBUFFER <0 | non-zero>

Top

Takes over keyboard and buffers up to 16 keys at a time if arg==non-zero, else turns OFF keyboard buffering and releases the keyboard.

Possible Errors

- INV MODE (214D) if run in "calculator mode" (only valid in running program).

KBDISKEY

Top

Returns 0 if no keys in buffer, else returns 1.

Possible Errors

- INV MODE (214D) if run in "calculator mode" (only valid in running program).

KBDKEY

Top

Returns the next keycode from the keyboard buffer.

Possible Errors

- INV MODE (214D) if run in "calculator mode" (only valid in running program).
- NULL DATA (149D) if no keys in buffer.

CRT Functionality

All of the following ‘CRT’ statements/functions apply ONLY to the ALPHA mode display.

CRTSETTOP row

Top

Sets CRT start row (0-63 on HP85, 0-53 (ALPHA) or 0-203 (ALPHALL) on HP87). This is the row# of CRT memory that appears on the top row of the CRT display.

Possible Errors

- none

CRTGETTOP

Top

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

Possible Errors

- none

CRTREAD$(row, column, len)

Top

Reads from CRT at (row,col) and returns a string of ‘len’ length from that location.

Possible Errors

- none

CRTWRITE string, row, column

Top

Writes ‘string’ to CRT at (row,col). ‘row’ is absolute in CRT memory, NOT relative to CRT SETTOP ‘row’, so you can write to CRT memory that is not currently displayed.

Possible Errors

- none

CRTROWS

Top

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

Possible Errors

- none

CRTCOLS

Top

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

Possible Errors

- none

CRTON <0 | non-zero>

Top

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

Possible Errors

- none

CRTCURSOR state,[row,col]

Top

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

Possible Errors

- none

CRTCURSCOL

Top

Returns cursor column number.

Possible Errors

- none

CRTCURSROW

Top

Returns absolute (NOT relative to TOP row) cursor row number.

Possible Errors

- none

Options/Settings

SDSLASH   0 | non-0

Top

Controls whether ‘/’ or \’\\’ is used in paths returned to the user.

- SDSLASH 0 sets it to ‘/’ (the default)
- SDSLASH 1 sets it to \’\\’.

This setting is remembered via an SD file over a power cycle.

Possible Errors

- none

SDSLASH$

Top

Return Value

- Returns a 1-character string containing either ‘/’ or \’\\’

Possible Errors

- none

SDEOL   0 | non-0

Top

Controls whether SDEOL$ returns LF or CR/LF as the EOL sequence.

(Cont'd)

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

This setting is remembered via an SD file over a power cycle.

Possible Errors

- none

SDEOL$

Top

Return Value

- Returns a 1-character or 2-character long string containing the current EOL sequence, either LF or CR/LF. Good for using in SPRINTF statements when writing text to an SD file.

Possible Errors

- none

Miscellaneous Commands

AUXREV

Top

Return Value

- Returns the AUX ROMs revision number (from ROM 361)

Possible Errors

- none

EBTKSREV$

Top

Return Value

- Returns a string containing a date and time of building the EBTKS firmware

Possible Errors

- none

RMIDLE

Top

This is not an actual command that can be typed at the Series 80 keyboard, it is used as a bidirectional channel between the Series 80 computer and EBTKS for tasks that involve passing data that is not a function of program execution, such as the EBTKS injecting characters into the keyboard stream, or intercepting keyboard character before the Series 80 computer "sees them"

more details later

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

Top

LED 1 is LED closest to the Power Inlet of the HP8x computer

LED 2 is LED closest to the EBTKS SD Card

First parameter:

1 Set LED 1

2 Set LED 2

3 Set both LEDs to the same color

Physically, LED 1 and 2 contain 3 internal LEDs and a controller integrated circuit (IC) within a 5 x 5 mm package, in the two back corners of the EBTKS circuit board. These 3 internal LEDs in each package emit light in either Red, Green, or Blue wavelength. By adjusting the brightness of these internal LEDs, a vast range of colors and brightnesses can be achieved. A web search for WS2812B or WS2812E datasheet can provide further technical information.

The SETLED command sends the R, G, B values to the IC, which in turn adjusts the intensity of the respective LEDs.

The LEDs can nominally show 16 million colors (a 24 bit value) but realistically very small changes in the control values will not be perceptible. This is particularly true at the brighter end of the brightness range. Each of the three parameters R, G, and B are numbers in the range 0 to 255. 0 is off, and 255 is full brightness for that color. Because the LEDs are not ideal, and they are not intensity matched, and are discrete with some gap between them (less than a mm), the mixing of the light is far from perfect and trying to get white light will be disappointing. Also the LEDs are quite bright and I find that values over 60 can be hard to look at directly. Your mileage may vary.

Possible Errors

- ARG OUT OF RANGE (system 11D) (if LED# is not 1,2, or 3)

BOOT

This command will re-boot your system, resetting both the Series80 computer and EBTKS. Any modified program that is in the Series80 computer memory will be lost, just like the SCRATCH keyword. Save your programs often when doing development.

LIF DISK FEATURES

RSECTOR   dstVar$, sector#, msus$

Top

Reads a raw 256-byte sector from the specified drive.

```
'dstVar$' must be dimensioned to at least 256 bytes (which is where
          the sector will be read into).
'sector#' is a number from 0 to NumSectors-1 (where NumSectors is
          the number of sectors available on the disk).
'msus$'   is the ":Dxxx" specification of the disk drive.
```

Possible Errors

- Typical Mass Storage ROM errors

WSECTOR   src$, sector#, msus$

Top

Writes a raw 256-byte sector to the specified drive.

> **WARNING!!! Be VERRRRRRRY careful using this, as you can easily trash your disk and lose ALL files stored on it!!!!**

Possible Errors

- Typical Mass Storage ROM errors

SDEXPORTLIF   LIFname$, SDname$ [, headerFlag]

Top

Exports a LIF file from a LIF disk to an SD file on the SD card. No translation of any sort is done on the contents of the file.

If the LIFname$ contains only an MSUS rather than a filename, then the ENTIRE disk is output, including volume and directory sectors and ALL files, resulting in an “emulated disk file” that can be used with the Series 80

(Cont'd)

emulator or with EBTKS on the SD card, or shared with others.

headerFlag:

> If 0, then no “LIF header” is written at the start of the SD file, and ONLY the ‘valid’ bytes from the LIF file are written. If non-0, then a 512-byte “LIF header” (on volume sector, one directory sector) is written to the file, followed by ALL the sectors from the LIF disk that were allocated to the file (which may include a bunch of ‘empty’ unused sectors).

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

Possible Errors

- INVALID MSUS (115D)
- FILE/PATH (217D) if disk is not LIF formatted
- FILE NAME (167D) if LIF file is not found
- FILE SIZE (223D) if the LIF file is >65536 bytes long

SDIMPORTLIF   LIFname$,   SDname$

Top

Single file version

Import a file on the SD Card named SDname$ that must be in LIF file format.

Such files may be created by the SDEXPORTLIF, but also the output of Everett Kaser’s ASM85 program. The file is stored on a LIF disk, which could be a physical disk, or an emulated disk provided by EBTKS. LIFname$ can include an msus$.

The LIF file format has a headerFlag=1 (i.e., with a “LIF header” included, containing volume and director ‘sectors’). The SD file MUST be prefaced with a “LIF header” (a 256-byte “Volume sector” and a 256-byte “Directory sector”).

Examples

- SDIMPORTLIF “MYPROG:D300”, “/SDDir/prog.bin”

  The imported BIN program created with ASM85 and in the directory SDDir on the SD Card will be stored on disk D300 with the name MYPROG and the CAT command will show that the type is BPGM

- SDIMPORTLIF “data”, “peoplesnames”

  The imported LIF file that is in the current working directory on the SD Card, named “peoplesnames” will be stored with the name “data” on the current MASS STORAGE IS drive. Remember, the imported file must be in LIF format, so probably originally written to the SD Card with the SDEXPORTLIF command

(Cont'd)

Whole Disk version

If the LIFname$ doesn't include a filename, but rather is just an MSUS$, then the SDname$ file is expected to be a full "emulated LIF disk," such as SDEXPORTLIF would create if it is run with just an MSUS for the LIFname$.

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

Possible Errors

- SD ERROR (213D) if error reading SD file
- INVALID MSUS (115D) if problem with LIF file name
- FILE/PATH (217D) if disk is not LIF formatted
- FILE NAME (167D) if SD file is not found
- FULL (MS ROM 128D) if the disk is too full

SDPATH$(index#, path$)

Top

Return Value

- Returns one element of path$
    - If index# is positive and non-zero, 1 returns 1st element, 2 returns 2nd, etc.
    - If index# is negative, -1 returns last element, -2 returns 2nd from last, etc.
    - If index# is ONE greater beyond the number of elements, "" (zero-len string) is returned.
    - Otherwise, error.

Possible Errors

> 11 ARG OUT OF RANGE (if index# is out of range)

SDCHAIN   name$

Top

Same as system CHAIN, just from SD file (created via SDSTORE).

Caution:

- There is no file type checking. If you SDLOAD a file that was not SDSTORE'd, chaos will be unleashed and the end of the universe will be nigh.

Possible Errors

- FILE/PATH (217D) error in file/path name, or error opening/creating the file

Top

SDSTOREBIN nameSD$

Stores the Binary Program currently in memory to an SD file. A “LIF header” is included on the front of the file. This is essentially a shortcut way of doing STOREBIN “filename” and then SDEXPORTLIF “filename”,”SDname”.

Possible Errors

- FILE/PATH (217D) error in file/path name, or error opening/creating the file
- SD ERROR (213D) if write failed to write everything

SDLOADBIN nameSD$

Top

Loads 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, chaos will be unleashed and the end of the universe will be nigh.

Possible Errors

- FILE/PATH (217D) error in file/path name, or error opening/creating the file

SDBATCH nameSD$

Top

Reads characters from nameSD$ and types them on the display. End-of- lines (CR/LF or LF) cause the END LINE key to be sent. Other Series 80 specific keys can be sent using the \ character followed by the three octal digits of the keycode.

The CRT may have text on the cursor line, rather than a clear screen. As you know, when you type a command, if there are characters on the line after the command you have typed, it will mess up your command. For HP85A/B, you must also make sure that the previous line does not have a character in the last position (character position 32) as that will cause the previous line to be treated as part of the command too. Here are two strategies that could work:

1. Clear the screen before entering commands using octal `\222` at the beginning of the character sequence
2. Clear the current line with “-line” code `\240` , then move down 1 line (`\242`) and clear that line as well (`\240`), then enter the desired command. So the beginning of “command” would be “ `\240\242\240` rest of the command”
   This sequence assumes the cursor is in column 1.

You don’t need to use `\232` for the end of line character, or `\042` for the double quote characters. Normal ends of lines in the batch file will be interpreted as the end line that the HP85 expects.

The contents of the batch file could be:

```
LOAD "LEDTEST-1"
RUN
```

(Cont'd)

Unlike the batch files on Windows OS systems, this batch capability is very simple. Just the sequential transfer of text.

Here is a link to a table of Octal Codes .

Possible Errors

- FILE/PATH (217D) error in file/path name, or error opening/creating the file

Other error messages

AUXROM has some internal functions that are called by other functions, and these internal functions can also return error codes

LOW-LEVEL FUNCTIONS

For Diagnostics and testing new ideas

These three Keywords can be used for diagnostic purposes to test the AUXROM code and also as hooks to write EBTKS routines without writing new AUXROM code.

For all three of these Keywords (AUXCMD, AUXBUF$, AUXOPT$)

- 0-16 bytes of opt$ get written to A.BOPT00-A.BOPT15
- buf$ gets written to A.BUFx where ‘x’ is buf#.
- A.BLENx gets set to the length of buf$.
- usage# gets written to A.BUSEx.
- MBx gets set to 1.
- x gets written to HEYEBTKS to send the command.

Note that the above memory references are in terms of the labels used in the AUXROM source code. The equivalent locations in EBTKS C code are as follows

```
A.BOPT00-A.BOPT15    AUXROM_RAM_Window.as_struct.AR_Opts[16]
A.BUFx               AUXROM_RAM_Window.as_struct.AR_Buffer_0
                                                AR_Buffer_1
                                                AR_Buffer_2
                                                AR_Buffer_3
                                                AR_Buffer_4
                                                AR_Buffer_5
                                                AR_Buffer_6
A.BLENx              AUXROM_RAM_Window.as_struct.AR_Lengths[8]
A.BUSEx              AUXROM_RAM_Window.as_struct.AR_Usages[8]
MBx                  AUXROM_RAM_Window.as_struct.AR_Mailboxes[32]
```

These are mostly of use to AUXROM and EBTKS devlopers.

Avoid use, except for these specific cases:

```
AUXCMD buf#, usage#, buf$ , opts$
AUXOPT$( buf#, usage#, buf$ , opts$)
AUXBUF$( buf#, usage#, buf$ , opts$)
```

AUXCMD   buf#, usage#, buf$, opts$

Top

How to use AUXCMD:

- A command is setup using the buffer (and matching mailbox and usage) specified by buf# (0..7)
- The related usage[buf#] is set to usage# (this is the function code)
- The string buf$ is copied to the designated buffer, and the related length values is set appropriately
- At most 16 bytes from the string opt$ are copied to AUXROM_RAM_Window.as_struct.AR_Opts[16]
- A call is made to EBTKS, specifying the buffer number buf#
- EBTKS processes the specified command (in usage[buf#]). There is no return value
- The appropriate usage location is set to 0 if there is no error, or an error code
- EBTKS indicates completion by setting the appropriate mailbox to 0

Possible Errors

- ARG OUT OF RANGE (system 11D)
- INVALID PARAMETER (system 89D)
- STRING OVERFLOW (system 56D)
- MEMORY OVERFLOW (system 19D)
- custom AUXROM msg (209D)

Possible Warning

- NULL DATA (system 7)

AUXOPT$(buf#, usage#, buf$, opt$)

Top

How to use AUXOPT$:

- A command is setup using the buffer (and matching mailbox and usage) specified by buf# (0..7)
- The related usage[buf#] is set to usage# (this is the function code)
- The string buf$ is copied to the designated buffer, and the related length values is set appropriately
- At most 16 bytes from the string opt$ are copied to AUXROM_RAM_Window.as_struct.AR_Opts[16]
- A call is made to EBTKS, specifying the buffer number buf#
- EBTKS processes the specified command (in usage[buf#]) and returns 16 bytes of data in AUXROM_RAM_Window.as_struct.AR_Opts[16]
- The appropriate usage location is set to 0 if there is no error, or an error code
- EBTKS indicates completion by setting the appropriate mailbox to 0
- The AUXROM code completes processing of AUXOPT$ by returning the options to the BASIC environment as the string result of making this call to AUXOPT$

(Cont'd)

Return Value

- A 16-byte string is returned containing the bytes of A.BOPT00-A.BOPT15.

Possible Errors

- ARG OUT OF RANGE (system 11D)
- INVALID PARAMETER (system 89D)
- STRING OVERFLOW (system 56D)
- MEMORY OVERFLOW (system 19D)
- custom AUXROM msg (209D)

Possible Warning

- NULL DATA (system 7)

AUXBUF$(buf#, usage#, buf$, opts$)

Top

How to use AUXBUF$:

- A command is setup using the buffer (and matching mailbox and usage) specified by buf# (0..7)
- The related usage[buf#] is set to usage# (this is the function code)
- The string buf$ is copied to the designated buffer, and the related length values is set appropriately
- At most 16 bytes from the string opt$ are copied to AUXROM_RAM_Window.as_struct.AR_Opts[16]
- A call is made to EBTKS, specifying the buffer number buf#
- EBTKS processes the specified command (in usage[buf#]) and returns a string in the same buffer that was use to pass buf$ to it. The appropriate length value is set to the length of the string.
- The appropriate usage location is set to 0 if there is no error, or an error code
- EBTKS indicates completion by setting the appropriate mailbox to 0
- The AUXROM code completes processing of AUXBUF$ by returning the string in the buffer to the BASIC environment as the string result of making this call to AUXBUF$

Return Value

- A string is returned containing the A.BLENx bytes of A.BUFx.

Possible Errors

- ARG OUT OF RANGE (system 11D)
- INVALID PARAMETER (system 89D)
- STRING OVERFLOW (system 56D)
- MEMORY OVERFLOW (system 19D)
- custom AUXROM msg (209D)

Possible Warning

- NULL DATA (system 7)

ADDING NEW FEATURES TO THE HP-85 AUX ROMS

ADDING NEW KEYWORDS

In 85aux1.src, find these tables:

- KEYWORDS ASCII keyword table
- TROM#S single-byte values of the ROM# in which each keyword's runtime code exists
- RUNTIM runtime routine table
- PARSES parse routine table

Find an appropriate open slot in these tables that:

- Isn't in use (ALL of the following must be true):
  - It has a '200' in the KEYWORDS table
  - It has a '0' (or is bsz'd) in the TROM#S table
  - It is bsz'd in both RUNTIM and PARSES tables
- Isn't marked RESERVED, *unless* you have to use one of those in order for your longer keyword to not be 'blocked' by an already existing shorter keyword. For example, the MOUNT keyword already exists. If you try to add the MOUNTAIN keyword AFTER the MOUNT keyword it will be 'blocked', the PARSER will never see the MOUNTAIN keyword, because it will find the MOUNT keyword first. Therefore, you'll need to use a RESERVED slot EARLIER in the table than the MOUNT keyword so that both MOUNTAIN and MOUNT will be spotted by the PARSER. This is the ONLY time you should use a keyword slot marked in the KEYWORDS table as RESERVED.

NOTE:

> Finding or making an open slot in the above tables MAY involve moving the 377 marker at the end of the ASP keywords in the KEYWORDS table. Is is CRITICAL that this table not get longer or shorter (in terms of the number of ENTRIES, not in terms of the number of BYTES), as there are tokens at the very end of this table that are "hidden (or worker) tokens" that are parsed into the token stream by an actual keyword (a particular example, at this point in time, are the three tokens 374, 375, and 376 which are used by the SPRINTF keyword to do "hidden things" at runtime). So, to keep from screwing those up, if you want to add a new keyword and need to move the 377 end-of-table marker, for each keyword you add, you must:
>
> - Delete the first 200 byte immediately AFTER the 377 marker
> - Insert an ASP statement immediately BEFORE the 377 marker
>
> You may NOT insert new keywords in between existing ones, as that would change the old keywords' TOKEN#, which would cause any programs written using an older version of the AUX ROM to stop working and likely crash, because the parsed tokens in that stored program would no longer match the token numbers in the new AUX ROM.

Once you've found your slot:

1. Add the ASCII keyword to the KEYWORDS with an ASP opcode.

(Cont'd)

2. Add the AUX ROM ROM#, in which the RUNTIME routine for the keyword is located, to the TROM#S table in the matching ‘slot’.

3. Add the address of the runtime routine in AUX ROM 1 to the RUNTIM table.

   For most STATEMENTS, this will be either “RUN1.” (if it’s a NON-programmable statement, i.e., a calculator-mode-only COMMAND) or “RUN2.” (if it’s a normal PROGRAMMABLE statement). The ‘1’ in “RUN1.” reminds you that the attribute is 141, whereas the ‘2’ in “RUN2.” reminds you that the attribute is 241. For most functions, there will probably already be an existing runtime entry point in AUX ROM 1 that has the right attributes for your function, and you can either use an already-existing label following those attributes, or add a new label. For example, if you’re adding a numeric function with no arguments, the attributes your runtime entry point needs are “0,55”, and searching through the function runtime entry points in AUX ROM 1, you’ll find that AUXERRN. has those attributes. So, you could either just use “AUXERRN.” as your runtime entry point, or add a new unique label immediately after the line containing the “AUXERRN.” label, so that the “0,55” attributes are immediately before both labels.

   If you can’t find an already existing function with the right attributes, then you’ll need to create a new entry. Many of these function entry points have no code and so will simply “fall through” the attributes of following functions. As long as those attribute bytes are all <200 (octal), they will be executed as ARP and/or DRP instructions, causing no damage, taking very little time, and saving bytes in the ROM. All of these functions eventually wind up at “RUN2.” at runtime, along with the normal STATEMENTS, and the “RUN2.” routine uses the TROM#S table to call the F_RUN function in the appropriate AUX ROM to actually execute the REAL runtime code for that keyword.

4. Add the address of the parse routine (if not a function) to the PARSES table. There are many ‘common’ parse routines already existing in the AUX ROM 1. If your new statement’s argument list matches the argument list of a previously created keyword, then you can just add a label for it and use the same parse routine. If you have special parsing needs, then you’ll need to add your own code, which may parse the whole statement (if it’s VERY brief) or call the actual PARSE routine in the other AUX ROM via FUNTAB.

5. In the other AUX ROM, where the actual runtime code for the keyword exists, be sure to add the address of the actual runtime routine to that AUX ROM’s RUNTIM table. See AUXROM2 and AUXROM3 for examples.

As an alternative to all of the above, I have found that just asking Everett to do it for me has worked quite well. PMF 11/27/2020

ADDING NEW NON-KEYWORD FEATURES

Some features that you might wish to add may involve taking over one or more system ‘hooks’. Also, some keywords you add may need to do some form of initialization at power-on or other times. The AUXROM 1, which contains all of the keyword and runtime/parse tables that the system can ‘see’ also contains the INIT routine. This init routine, once it has done anything it needs to do, calls an INIT routine in each of the sub-AUXROMs, to give them a chance to do whatever. It does this by calling the function in each AUX ROM’s FUNTAB (see F_INIT in 85auxdef.src and the FUNTAB in each AUXROM, the first entry of which is usually “def R_INIT”. It’s CRITICAL that you maintain the exact order and value of all of the F_ equates in 85auxdef.src, and construct/edit the FUNTAB tables accordingly, or the Holy Howling Hounds of Hades will invade your workspace.

Octal Keycodes for Special Keys on HP 85 A/B Keyboard

Key	Octal	Key	Octal	Key	Octal	Key	Octal
K1	200	INIT	214	-unused-	230	-CHR	244
K2	201	RUN	215	BKSPACE	231	HOMECURS	245
K3	202	PAUSE	216	ENDLINE	232	RESULT	246
K4	203	CONT	217	FASTBKSPACE	233	-unused-	247
K5	204	STEP	220	LFCURS	234	DELETE	250
K6	205	TEST	221	RTCURS	235	STORE	251
K7	206	CLEAR	222	ROLLUP	236	LOAD	252
K8	207	GRAPH	223	ROLLDN	237	-unused-	253
REW	210	LIST	224	-LINE	240	AUTO	254
COPY	211	PLIST	225	UPCURS	241	SCRATCH	255
PAPADV	212	KEYLABEL	226	DNCURS	242		
RESET	213	-unused-	227	INS/RPL	243		

Octal Keycodes for Special Keys on HP 86 and 87 Keyboard

Key	Octal	Key	Octal	Key	Octal	Key	Octal
K1	200	INIT	214	HCURS	230	DOWN CURSOR	244
K2	201	RUN	215	BKSPACE	231	K12	245
K3	202	PAUSE	216	ENDLINE	232	RESULT	246
K4	203	CONT	217	FASTBKSPACE	233	-unused-	247
K8	204	STEP	220	K7	234	A/G	250
K9	205	ROLL up	221	-LINE	235	ROLL down	251
K10	206	TEST	222	INS/RPL	236	RIGHT CURSOR	252
K11	207	K14	223	LEFT CURSOR	237	-unused-	253
-CHAR	210	LIST	224	E	240	K13	254
CLEAR	211	PLIST	225	K5	241	TRACE/NORMAL	255
-unused-	212	KEYLABEL	226	K6	242		
RESET	213	-unused-	227	UP CURSOR	243		

EBTKS Downloads

Standard File Set for the SD card

Definition:
A **LIF disk image** is a file on the MicroSD Card, typically in the /disks directory, and with a file extension of .dsk
It represents either a floppy disk (size is 264 kB) or a 5 MB Winchester disk (size is 4743 kB). When you read or write to disks that are emulated by EBTKS, these are the files that represent those disks. Internally, these files have a directory and the files you see with a CAT command. For example, if the CONFIG.TXT file contains the following section:

```
{
  "Comment": "msus$ 300",
  "unit": 0,
  "filepath": "/disks/EBTKS_1.0_85.dsk",
  "writeProtect": false,
  "enable": true
},
```

EBTKS_1.0_85.dsk is the LIF disk image that is seen as :D300 , which is the default mass storage device with EBTKS. When you type CAT, you are getting the catalog (directory listing) from inside this file. When you load a program from this “disk”, you are actually reading the program from somewhere within this file, as indicated by the internal directory information. Similarly, if you are saving a program or writing data to :D300 (the default mass storage device) you are actually modifying this file.

Warning:
If you have been developing program and/or writing data to LIF disk images on your MicroSD Card, these should be saved before overwriting with these downloaded MicroSD Card file sets. See above definition.

Update V1.0.2:   Do I need it?

The primary purpose of release V1.0.2 is to fix a bug that affects some HP86 and HP87 computers. The bug is due to an unintended interaction between EBTKS and the Graphics ROM (001) in these systems. The HP85A/B do not have this ROM.

If you are not running programs that display text on CRT when in Graphics mode, the bug will not be seen. If you run such programs, you may still not see the bug, depending on the version of the Graphics ROM. Here is a trivial test program:

```
GRAPH @ MOVE 50,50 @ LABEL "HELLO WORLD!"
```

(Cont'd)

If the text is displayed correctly, your system is not affected by the bug.

Regardless of whether this bug affects your computer, this release contains some other updates that you might find useful. See below

If your HP86/87 displays the bug, and you don’t want to do a full update of the SD Card, See below for instructions to just do the minimal update.

Update V1.0.2

There are 2 MicroSD Card file sets available for download. One for HP85A/B, and the other for HP86/87. These are a complete distribution file set, equivalent the SD Card contents when you originally received your EBTKS. The contents of the two file set are close to identical, with the following differences:

- The CONFIG.TXT file in the root directory is copied from a different file, that is also in the root directory
    - For HP85A, it is a copy of CONFIG.TXT_for_HP85A_Tape_Drive_disabled
    - For HP87, it is a copy of CONFIG.TXT_for_HP87-32KB

- The pre-configured LIF disk images for :D300, :D301 and :D320 are different.
    - For HP85A/B
        - :D300 is associated with /disks/EBTKS_1.0_85.dsk
        - :D301 is associated with /disks/85Games1.dsk
        - :D320 is associated with /disks/85Games2.dsk

    - For HP86/87
        - :D300 is associated with /disks/EBTKS_1.0_87.dsk
        - :D301 is associated with /disks/87_Action_Games.dsk
        - :D320 is associated with /disks/87_Galaxy_Patrol.dsk

Please don’t just download and start editing the CONFIG.TXT file. Spend a little time reading the documentation, and save a whole lot of frustration Working with CONFIG.TXT

The normal way to use an update like this is as follows:

- Consider making a backup of your current SD Card, just in case you have any issues.
- Unzip the downloaded update in to a temporary directory on your PC/MAC/etc
- Make any needed changes to the CONFIG.TXT file
- From your existing SD Card, copy any LIF disk images that you have modified to the /disks directory within the temporary directory on your PC/MAC/etc
- Make any other needed changes.
- Copy all the files in the temporary directory to an SD Card. If you have more than one SD Card, maybe use a different one from your existing SD Card for EBTKS.
- Put the newly created SD Card in to EBTKS.

The downloaded .ZIP file should be un-zipped into a temporary directory. There are 8 CONFIG.TXT files provided, with the rest of each file name indicating its specific pre-configuration. Copy the appropriate one to CONFIG.TXT (you will need to delete or rename the default CONFIG.TXT that is provided)

The CONFIG.TXT file should be inspected to make sure that it matches your Series80 computer, and your desired ROMs and LIF disk images that are configured in the “hpib” section are what you want.

(Cont'd)

On your existing SD Card, if you have been STORE-ing programs or data, then you have modified the related LIF files in the /disks directory. You should make a copy of these files, and add them to the /disks directory in the temporary directory where you are building the new SD Card file set for your system.

The complete set of files in the temporary directory should be copied to a blank MicroSD Card, or the existing one (which hopefully you have made a backup).

What has changed in microSD Card version 1.0.2

- Updated all CONFIG.TXT files in the root directory to be more specific to each model of Series80 computer, and some minor errors related to default ROMs and RAM size for HP86B. There are now 8 example CONFIG.TXT files.
- For HP86 and 87, virtual disks :D301 and :D320 now point to disk images of HP86/87 games, some/all authored by Everett Kaser.
    - Note: These games have not been tested by the author of this page. If you find that to run them you need to follow some setup, like changing the MASS STORAGE IS setting, etc…, please email Philip so this page can be updated.
- Major improvement to the HELP system for HP85A/B, thanks to Martin Hepperle. He has authored 192 help articles that cover the ROM keywords for
    - I/O ROM
    - Advanced Programming ROM
    - Matrix ROM
    - Printer/Plotter ROM

  You too can see your name here by adding to the HELP system by writing additional pages. They are easy to do, and given the limits of the 32 character wide screen and maximum of 64 lines of text, terse prose is an advantage. If you feel inspired, co-ordinate with Everett to avoid duplicated effort
- Although not previously documented, in the directory tree starting at /EK_Disks , there about LIF 100 floppy disk images (any file that is 264 kB) that cover most of the PACs published by HP, and some other assorted material. To use these LIF disk images
    - Make a copy of the file and place it in the /disks directory
    - Add a file name extension of **.dsk**
    - Edit your CONFIG.TXT file to assign it to one of the existing MSUS definitions, or add a new one
- In the directory /other_disks_85 is:
    - Waveform Analysis PAC for HP85
- In the directory /other_disks_86_87
    - Waveform Analysis PAC for HP86 and 87
    - Circuit Analysis PAC for HP86 and 87
    - Math PAC for HP86 and 87
- The above two directories have a subdirectory with some documentation PDFs.

MicroSD Card file set as a ZIP for HP85A like computers

This SD Card File Set is intended to work with the following Series80 Computers

HP83
HP9915A
HP85A
HP85AEMC
HP85B
HP9915B

SD_Card_Image_V1.0.2_for_HP85A_and_B_2022_01_21.zip

Release 1.0.2 , January 21th 2022

MicroSD Card file set as a ZIP for HP86A/B and HP87 and 87XM computers

This SD Card File Set is intended to work with the following Series80 Computers

HP86A
HP86B
HP87
HP87XM

SD_Card_Image_V1.0.2_for_HP86_and_87_2022_01_21.zip

Release 1.0.2 , January 21th 2022

Minimal update for for Graphics Text bug

If your HP86/87 displays the Text bug described above, the minimum update is to download the file set for HP86/87 and copy the following 8 files from the **/roms87** directory of the download to the same named directory on your SD Card.

- rom361
- rom361.lst
- rom362
- rom362.lst
- rom363
- rom363.lst
- rom364
- rom364.lst

Firmware for Teensy 4.1

Release 1.0.0 , July 12th 2021

EBTKS_Firmware_2021_07_12_V1.0.0.hex

This firmware is intended to work with all Series80 Computers

After downloading the EBTKS Firmware, proceed to Updating the EBTKS Firmware

CONFIG.TXT

If you are downloading one of the SD Card images, it already has all of the following example CONFIG.TXT files in the root directory.

The following downloads are if you have messed things up and you need a clean copy to get something that works.

All of these are from the V1.0.2 update released of 21 January 2022

CONFIG.TXT_for_HP85A_Tape_Drive_disabled

CONFIG.TXT_for_HP85A_with_Tape_Drive

CONFIG.TXT_for_HP85B_Tape_Drive_disabled

CONFIG.TXT_for_HP85B_with_Tape_Drive

CONFIG.TXT_for_HP86A-64KB

CONFIG.TXT_for_HP86B-128KB

CONFIG.TXT_for_HP87-32KB

CONFIG.TXT_for_HP87XM-128KB

After downloading one of these CONFIG files, rename it to CONFIG.TXT . Depending on your system configuration, make any necessary edits to match your requirements, then copy it to the root directory of the SD Card. Detailed instruction for editing CONFIG.TXT can be found here Working with CONFIG.TXT

EBTKS Quick Reference Guide (QRG)

Martin Hepperle has created a compact guide to all the new keywords that are implemented by the AUXROMS on EBTKS.

Download the QRG

SanDisk Brand SD Card Datasheet

SDCard Datasheet

Github

EBTKS Teensy 4.1 Source Code on Github

EBTKS Schematic

V2.0 Schematic (Used for development and Beta Test)

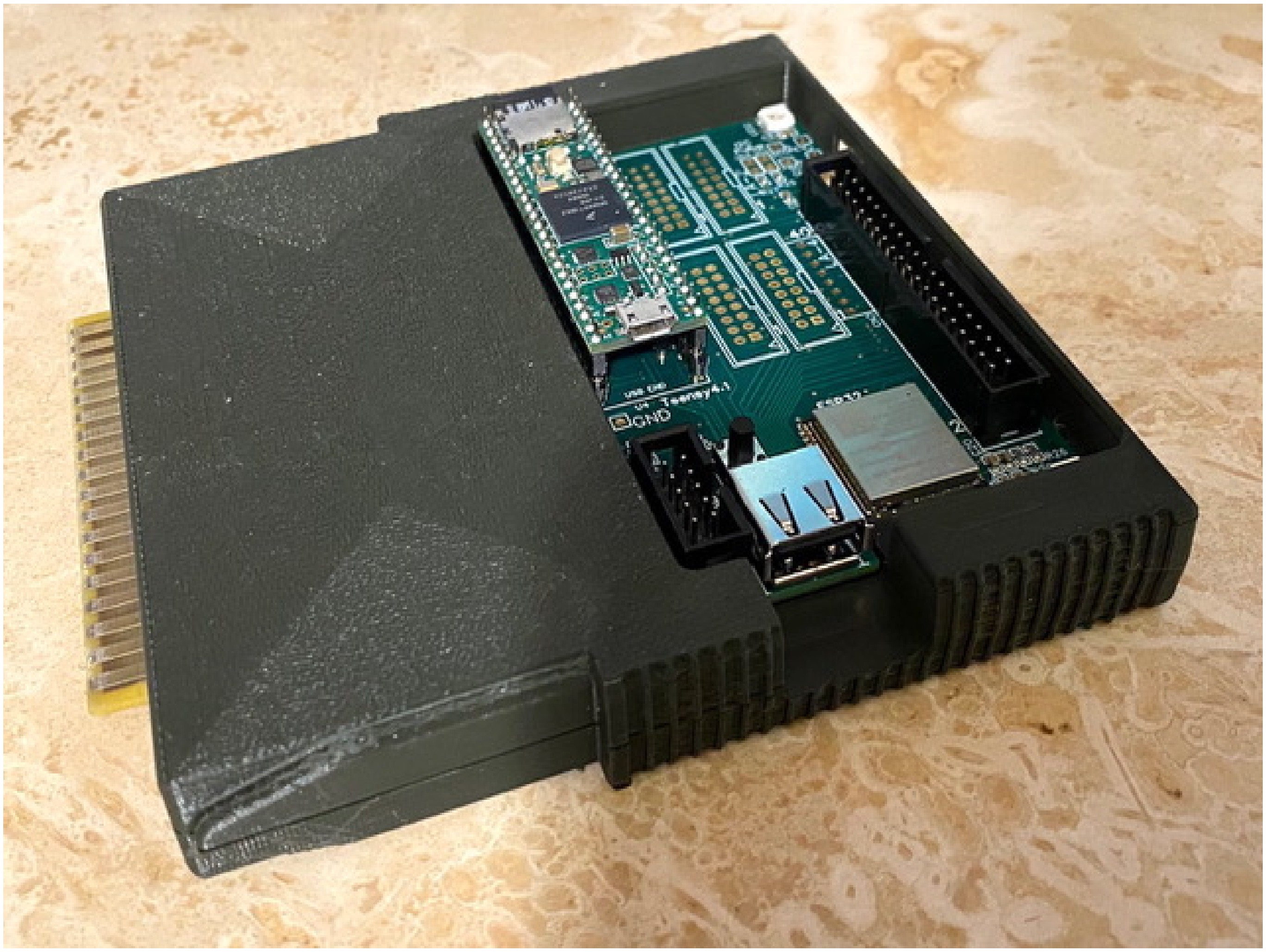
V3.0 Schematic (Production)

All EBTKS with Serial numbers of the form 01-0xx, 02-0xx, 03-0xx, 04-0xx, 05-0xx, 06-0xx are V3.0 Schematic, with no rework or modifications. All are ROHS compliant. All are a mix of automated assembly for surface mounted parts and hand assembly both at the factory, and by Philip (the Teensy 4.1 sockets and pins)

EBTKS 3D Printable cases

A few EBTKS users have designed 3D printable cases. This page is an index of these designs, and links to a separate page for each design which include additional images, and the files needed to 3D print the case. Some of these pages also include the design files which could be used as a starting point for a modification of your own.

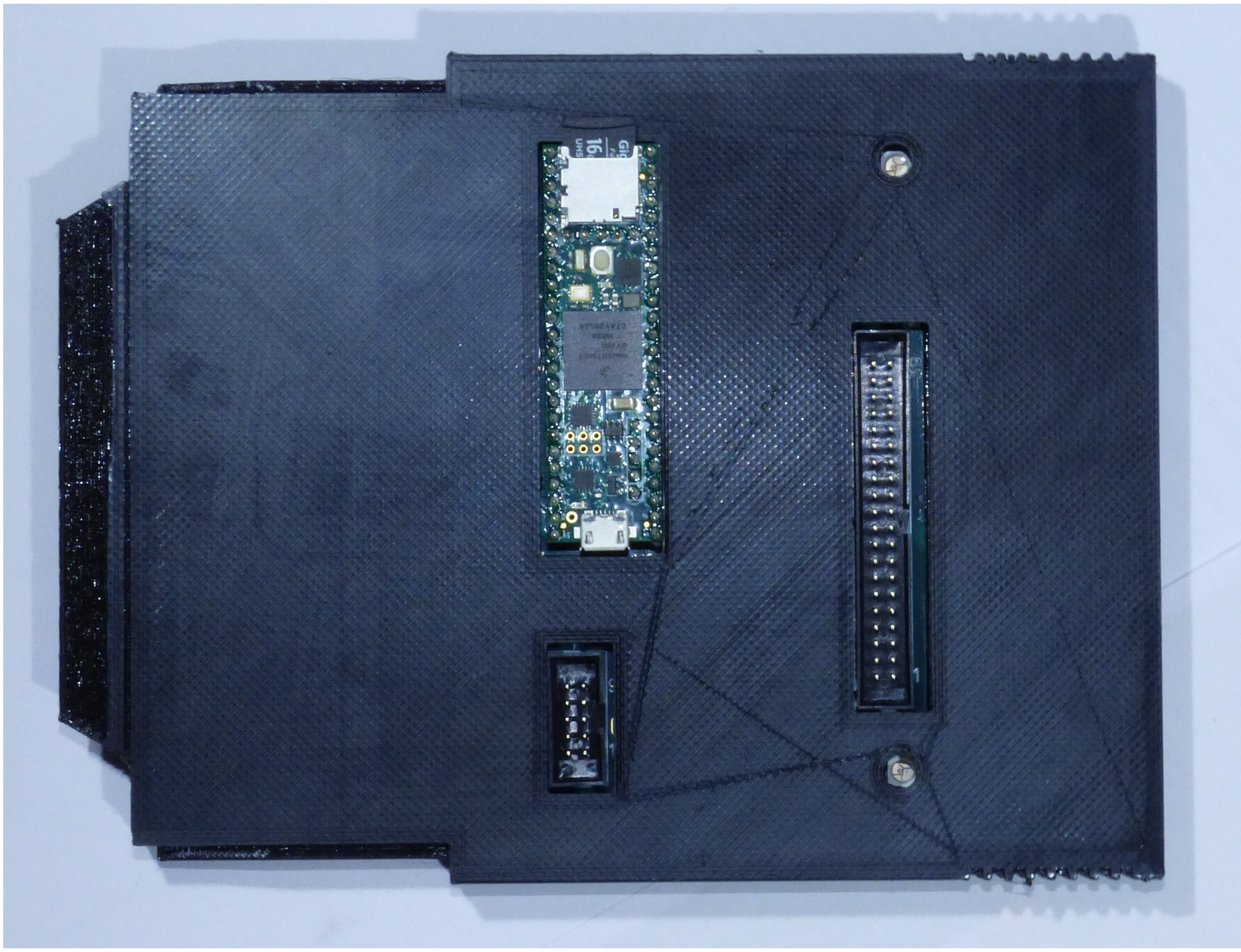
Daniel Simpson



Daniel's 3D case, Top View

More images, STL files and 3D print service

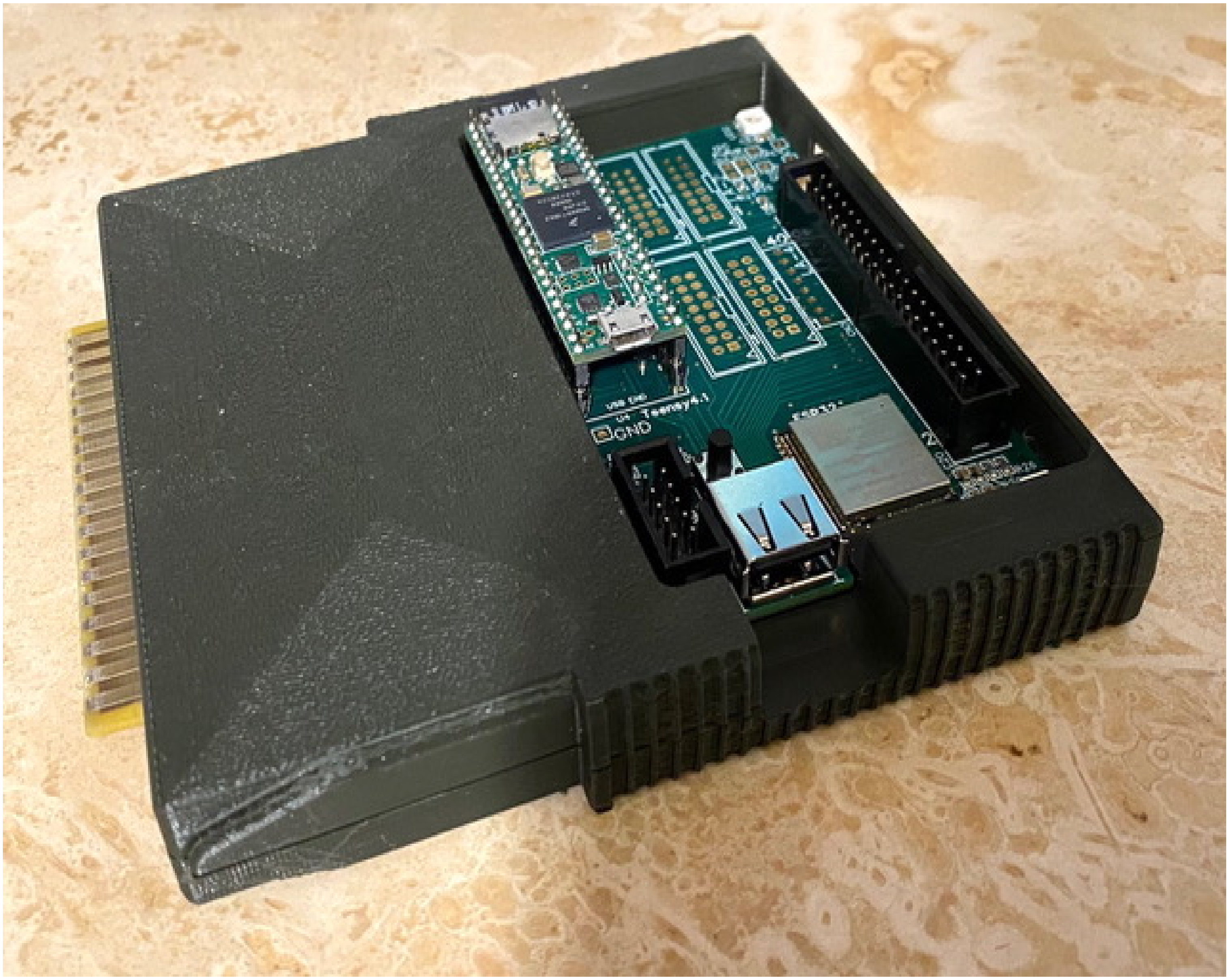
Martin Hepperle



Martin’s 3D case, Top View

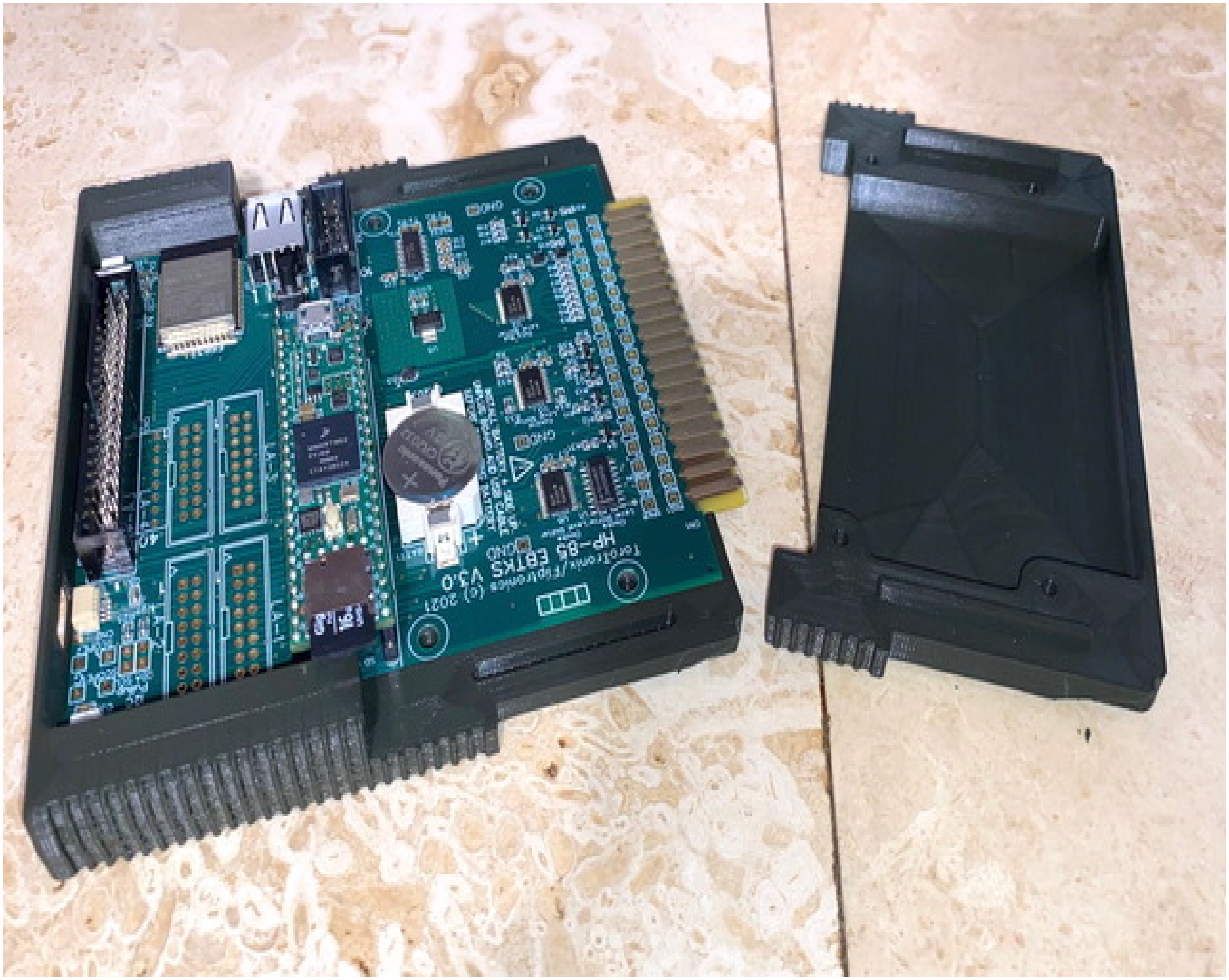
More images and STL files

Daniel Simpson’s 3D case

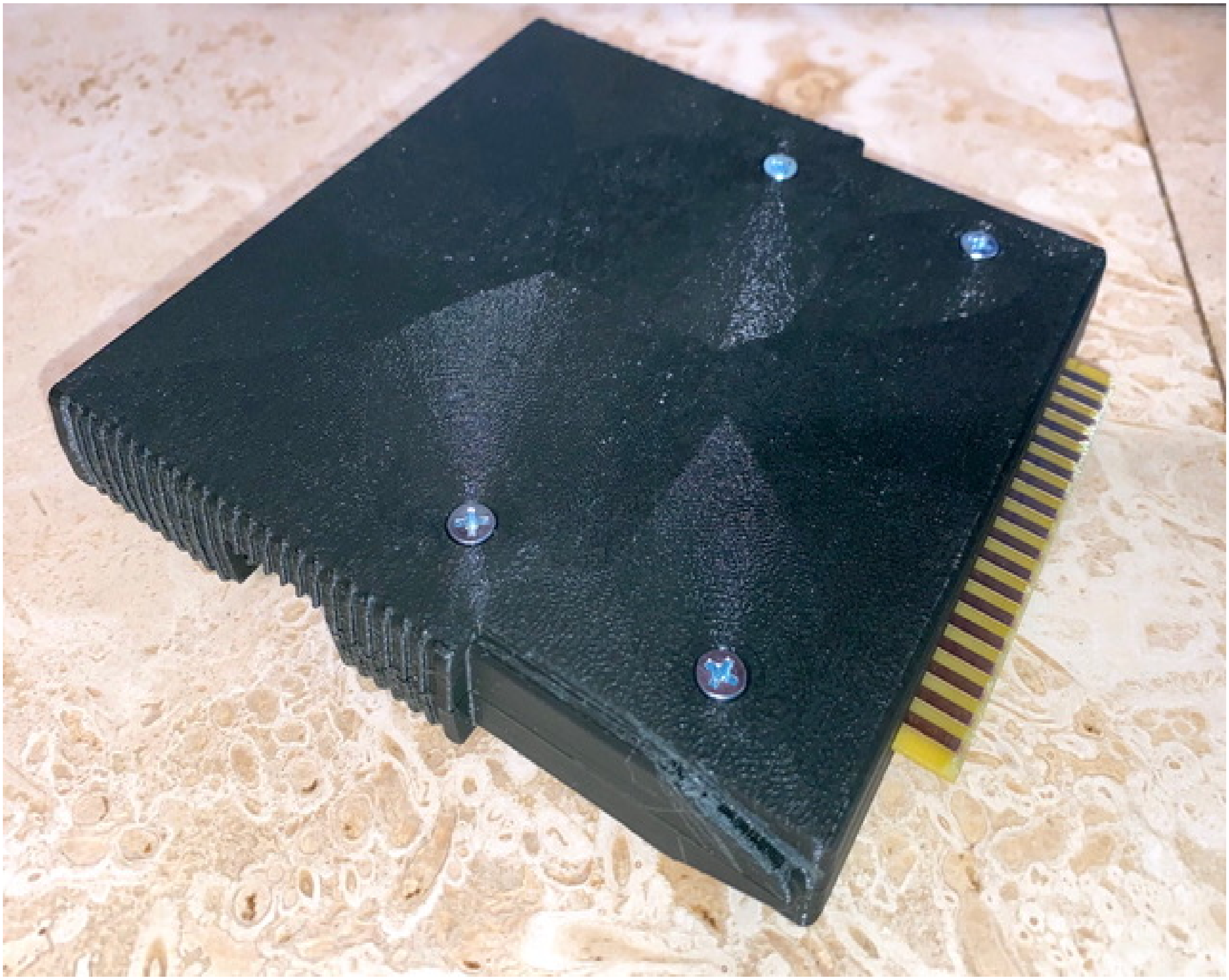


Daniel’s 3D case, Top

Daniel’s 3D case, Back



Daniel’s 3D case, Cover open



Daniel’s 3D case, Bottom

Download Links

Daniel_Simpson_3D_EBTKS_Bottom_Half.stl (Updated 2021_10_03)

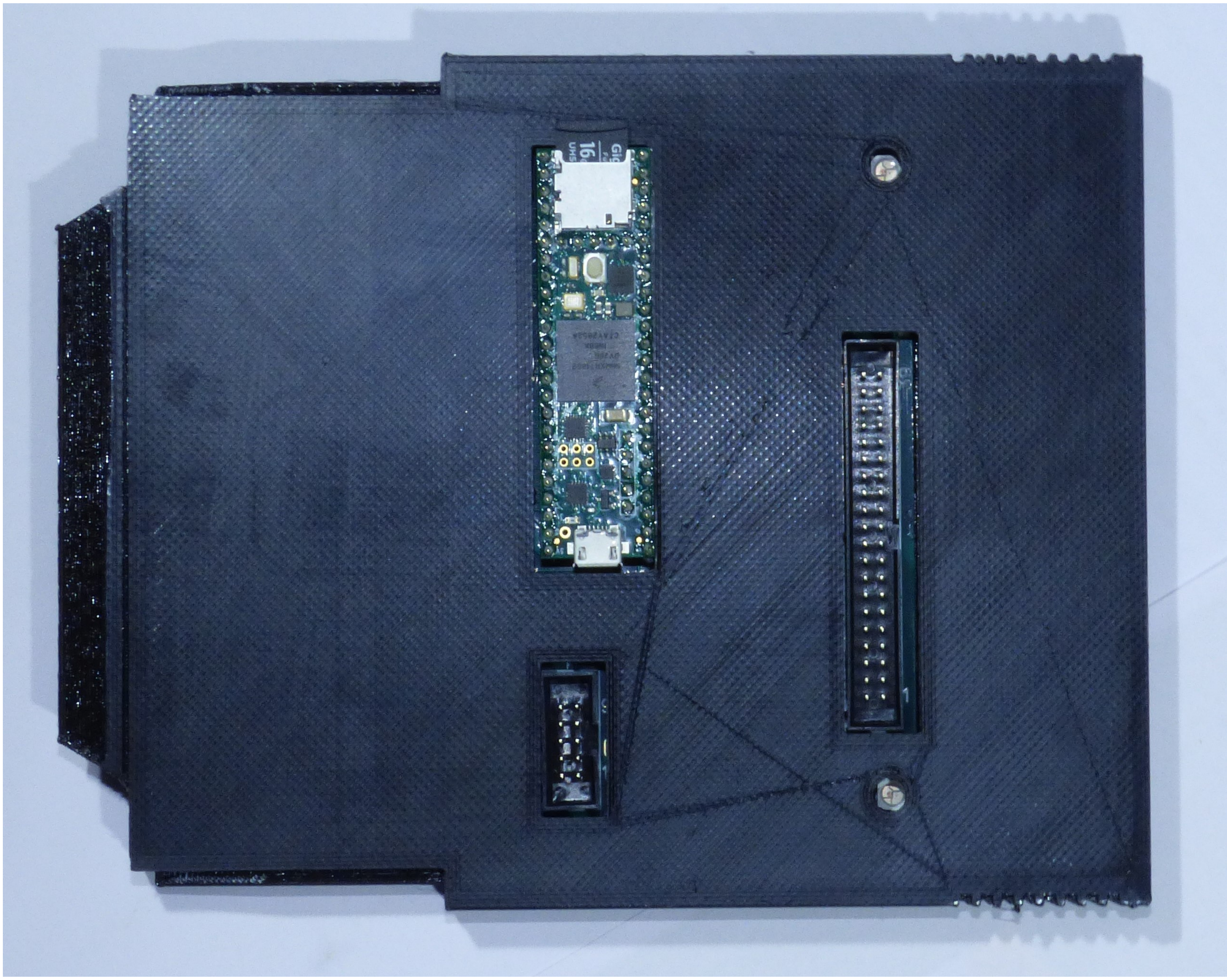
Daniel_Simpson_3D_EBTKS_Top_Half.stl

3D Printing Service

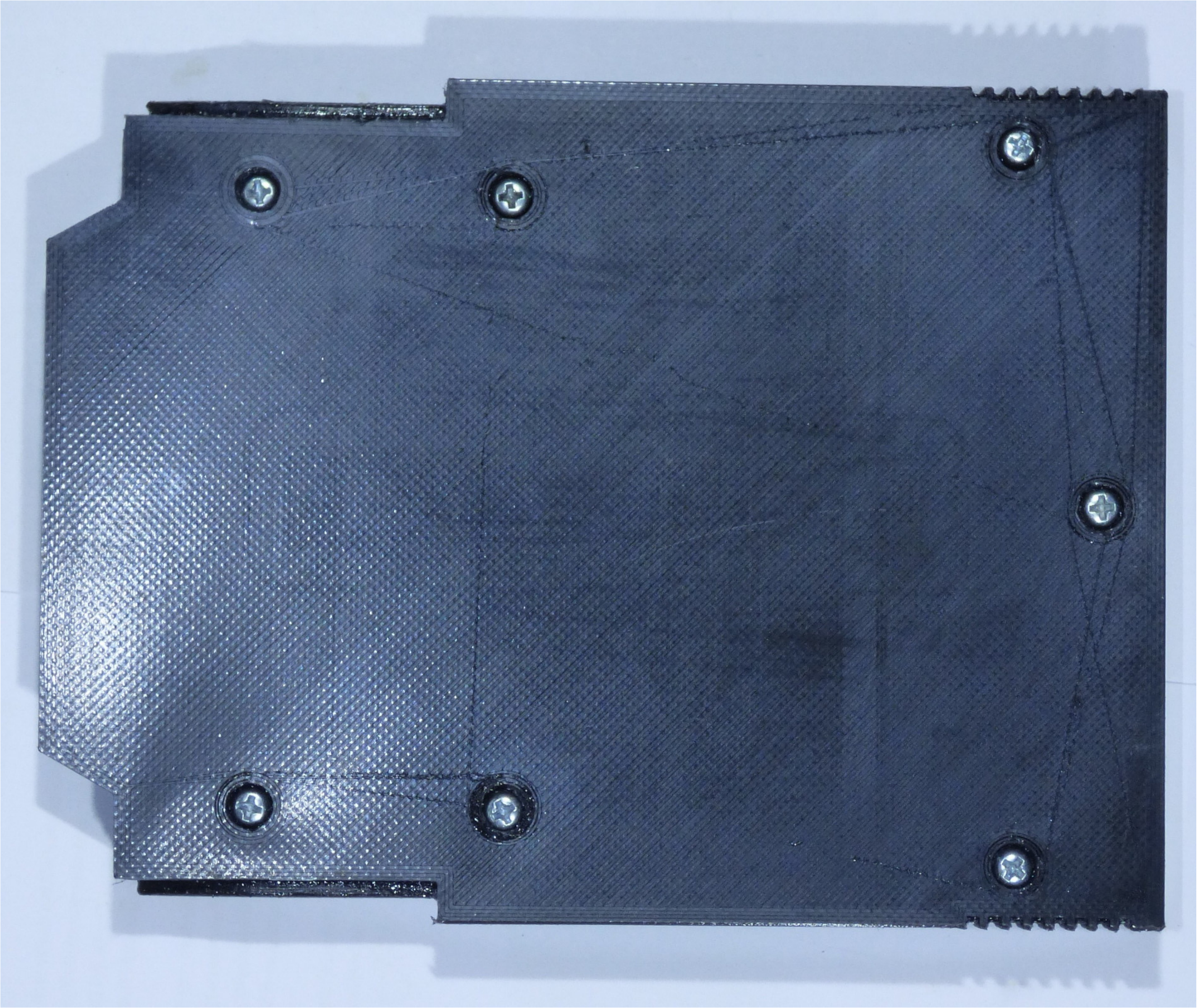
Daniel 3D prints calculator stands for many classic HP Calculators, and has an online store for ordering. He has now added the 3D case featured on this page.

Daniel's 3D print store

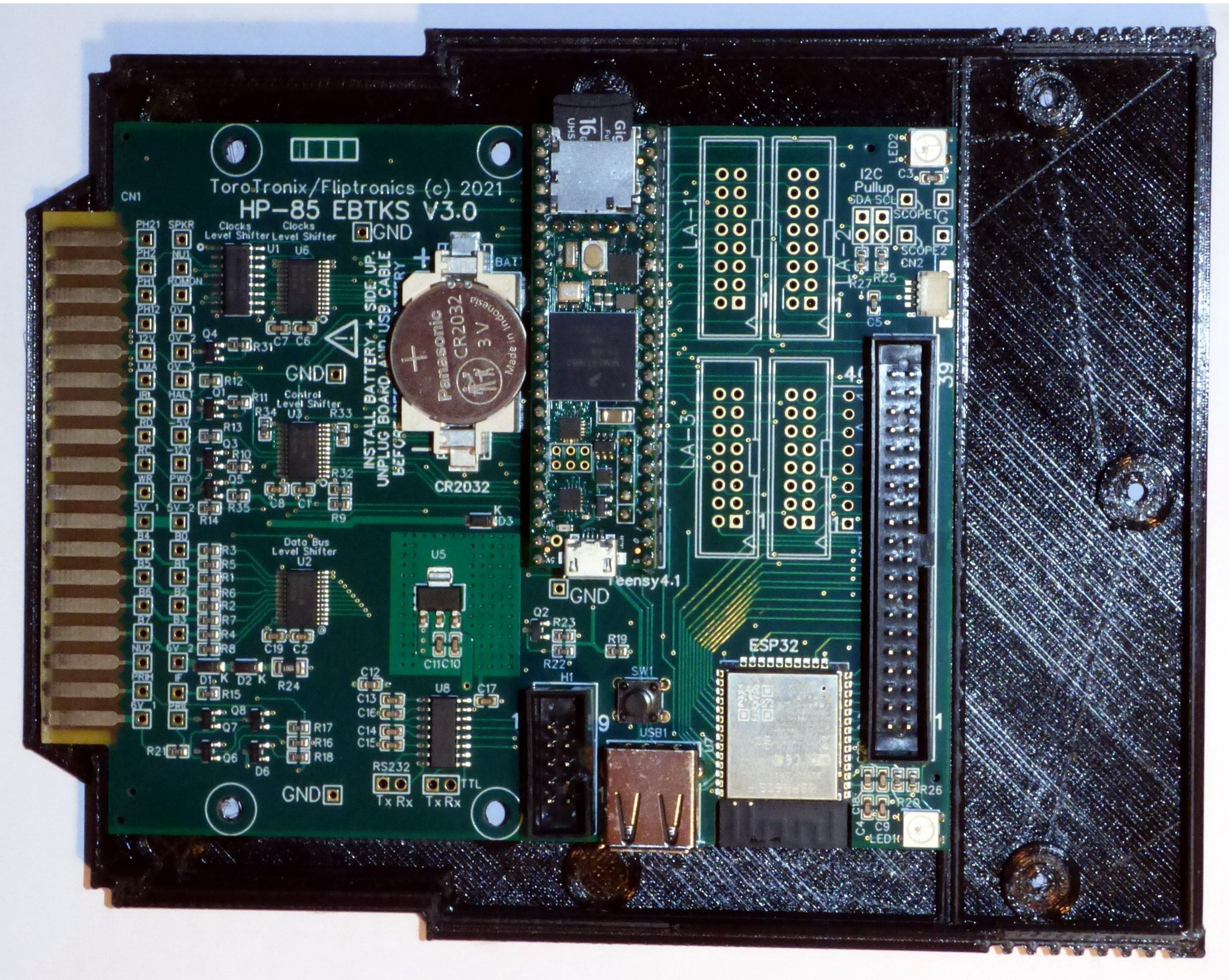
Martin Hepperle's 3D case



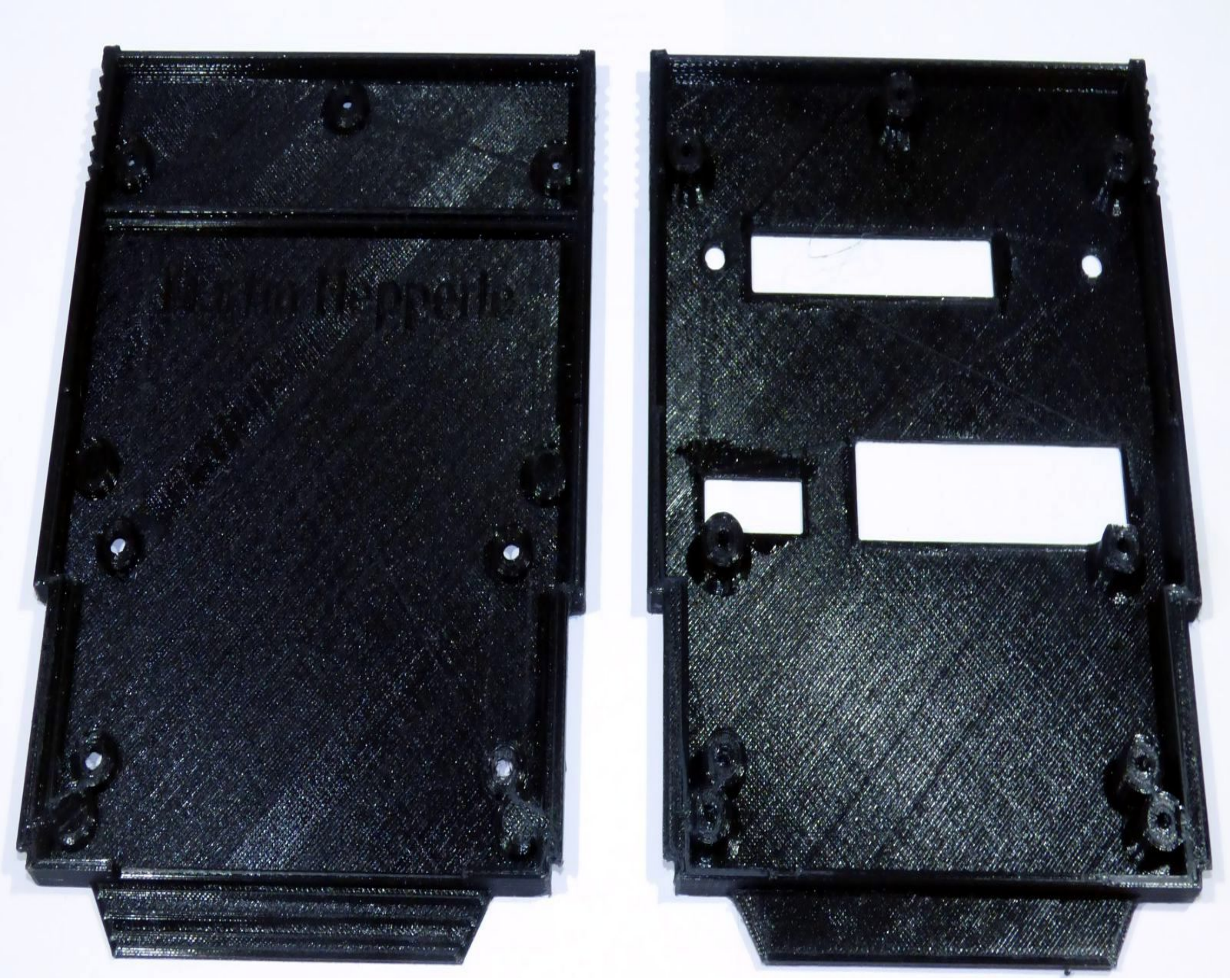
Martin's 3D case, Top



Martin’s 3D case, Bottom



Martin’s 3D case, Cover open



Martin’s 3D case, Top and Bottom

Martin’s 3D case, Close up on Teensy 4.1

Download Links

Martin's STL files are hosted on github

Updating the EBTKS Firmware

Notes:

EBTKS Firmware is software that is stored on a Flash chip that is part of the Teensy 4.1 module. This is separate storage from the SD Card that plugs into the Teensy 4.1 module, and holds the file system, emulated tapes and disks, and the Series80 Option ROMs.

This page documents the process for updating the Firmware.

If you are also updating the SD Card contents, that is done by plugging the SD Card into your computer (you may need an adapter from SD Card to USB). The SD Card should be seen as a FAT32 file system, and normal system commands can be used to copy files to and from the SD Card.

> If you are using the Micro-B USB port to connect to a terminal emulator to access the diagnostic channel, this connection must be closed for the firmware update process

Teensy 4.1

The core of EBTKS is a Teensy 4.1 from PJRC. A utility program called the Teensy Downloader is used to update FLASH memory on Teensy with the operational firmware. When new features are added to EBTKS or bugs are fixed, a new firmware file will be available on the Downloads page .

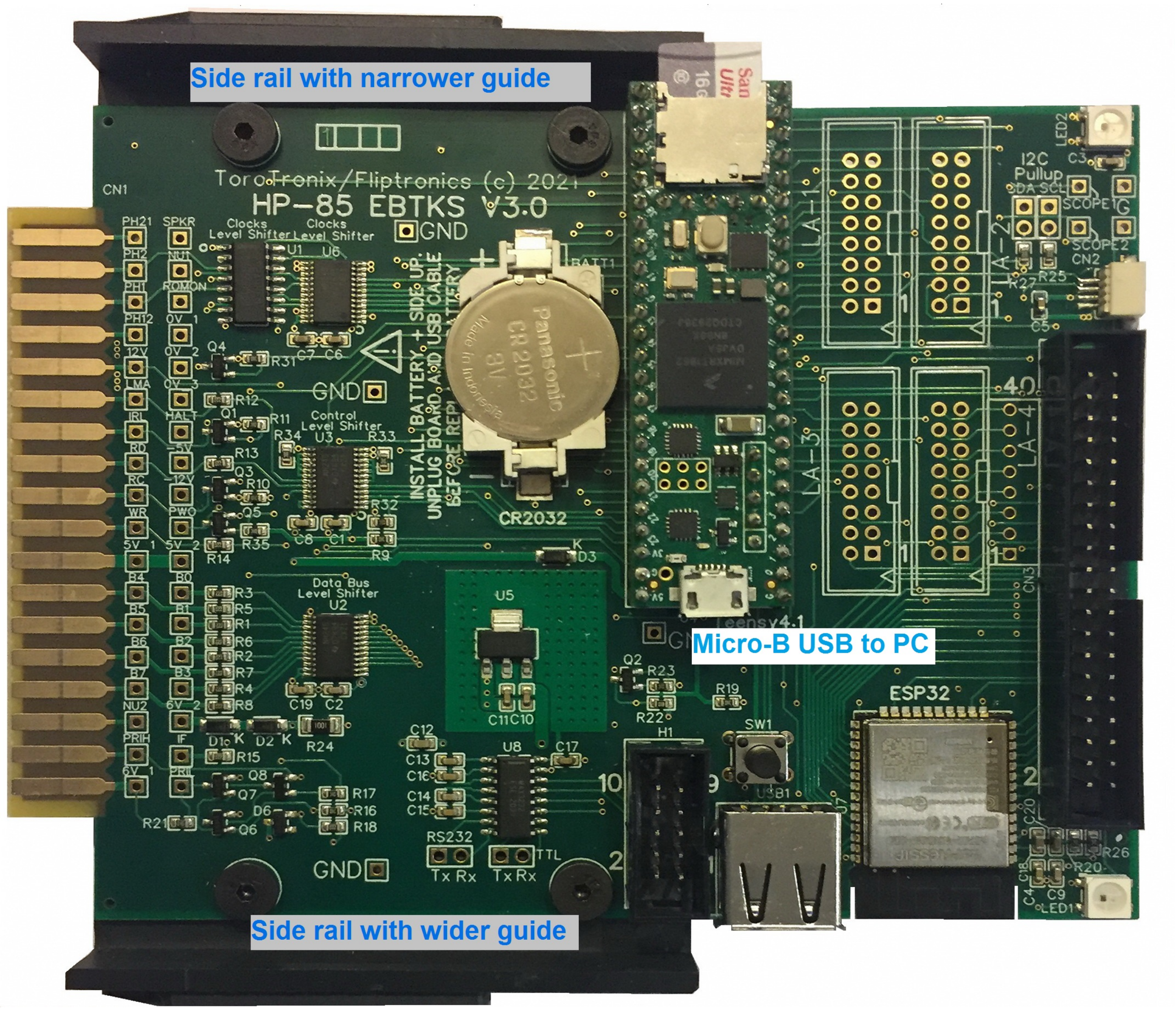
The firmware files have a file extension of .HEX

Procedures for doing the update are described below, for Windows, Linux, and Mac computers.

You will need a USB cable that on one end plugs into your computer (typically a USB-A plug), and the other end must have a Micro-B USB plug, that matches the Micro-B USB socket that is at one end of Teensy.

There is a USB-A socket on the main EBTKS board (bottom edge, near the middle in the picture below). It is not involved in the update process. Please ignore it.

(Cont'd)


Side rail with narrower guide

Micro-B USB to PC

Side rail with wider guide

Follow this link to the Teensy Downloader and follow the install instructions for your computer. It is available for Windows, Linux and Mac computers.

As I only have a Windows PC for doing downloads, I will describe how the firmware update is done with my computer. Hopefully others with either Linux or MAC will document their experience on their systems, and send them to Philip, to update this section of the documentation to cover these other systems

Windows

On the Windows page for Teensy Loader there are two downloads:

Teensy Loader Program
LED Blink, Both Slow & Fast

In all of the following description, the references to the white button are to the physical button on the Teensy, not the image on the Teensy.exe program, which is for a different model of Teensy.

Windows Short Form

- Download Teensy.EXE and the test blink programs in a ZIP file
- Connect the Teensy on EBTKS to your computer with a USB cable to the Micro-USB Connector
- Load and run the Fast Blink .HEX program for Teensy 4.1
- Put Teensy.EXE into Auto mode
- Load and run the Slow Blink .HEX program for Teensy 4.1
- Download firmware for EBTKS and program it into the Teensy 4.1

Windows Long Form

The Teensy downloader program is supplied as Teensy.exe , there is no setup process as the downloaded program is self contained and ready to run. Store it in an appropriate directory on your Windows PC, and if you want, create a desktop link as you would for any other runnable program.

As shown on the download page on the PJRC website for Teensy.exe , the program has a very simple user interface. Here is how to use it with EBTKS:

The Teensy needs to have a connection to your computer from the Micro- USB connector on the Teensy 4.1 , see above for the location. After making the connection for the first time, you may have to wait about a minute for the Windows operating system to retrieve the appropriate device drivers. This mean that your computer must be connected to the Internet.

The "LED Blink, Both Slow & Fast" test is a .ZIP file. Unpack the .Zip, it will put all the files in a directory named blink_both. Follow the instruction to check that Teensy.exe works on your system by connecting the Teensy on your EBTKS board via the Micro-USB connector to your computer. Teesny can remain plugged into the EBTKS board, and EBTKS can be either plugged into the HP85 with power on or off, or EBTKS can just be sitting on your desk. It is powered via the USB cable.

There are 38 files in the unpacked blink_both directory. The only two that you will be using are:
blink_fast_Teensy41.hex
blink_slow_Teensy41.hex

At the step where you are instructed to push the white button on Teensy, a red LED will turn on. It is near the Micro-USB connector. Soon after the red LED comes on the Teensy.exe program will turn the two arrow icons green. In the Teensy.exe program select "File", and navigate to the first test program "blink_both\blink_fast_Teensy41.hex". The left green arrow which points down is the program icon. When you click it, the red LED will get brighter for about a 1/4 of a second during the programming operation. When this step is complete the brightness of the red LED is lowered to its initial brightness.

Now click the green arrow that points to the right. This causes the Teensy to reboot and run the fast blinking program. The red LED turns off. Now click the gray-ed out "Auto button". It should turn green. Go to the "File" menu, and select blink_slow_Teensy41.hex . Press the white button on the Teensy and it should automatically program and reboot the Teensy. Sometimes the white button press needs to be done more than once.
If all the above was successful, you can now proceed to program the Teensy with the EBTKS firmware, which you can download from Downloads page .

Linux

Notes from Mike G. (thanks Mike). Please read all of this section before starting the update process:

- This is what I did on my Ubuntu 20.04LTS system to update the firmware on EBTKS
- The Teensy flash utility can be found here: https://www.pjrc.com/teensy/loader_linux.html as a tar.gz file
  There are two versions
  - Download Teensy Program (32 bit)
  - Download Teensy Program (64 bit)

  Download the appropriate one for your system
- Open a terminal window and change directory to the folder containing the downloaded tar.gz file
- Extract the file with tar, then copy the 00-teensy.rules file to the directory shown:
  - tar xf teensy_linux64.tar.gz
  - sudo cp 00-teensy.rules /etc/udev/rules.d/
- Reload udev and is required when making config changes
  - sudo udevadm control –reload
- Connect a USB cable between the Micro-B USB connector on Teensy and your computer
  - Mike G wrote that he unplugged the Teensy from EBTKS. Philip believes this should not be necessary. Since the pins on Teensy are delicate, avoiding unplugging it from EBTKS is desireable. Also, to avoid a possible issue with unpowered USB hubs, and performance limitations with some USB hubs, don’t connect via a USB hub. Connect directly to a USB port on your computer. Per the note at the top of this page, if you have been using a terminal emulator via this USB port on Teensy, this connection must be closed. Before unplugging the Teensy module, please try the following in the order presented, and report back to Philip what your experience is, so that this documentation can be updated to the least complex procedure.

  1. EBTKS plugged into your Series80 computer with the power on
  2. EBTKS plugged into your Series80 computer with the power off
  3. EBTKS unplugged from your Series80 computer
  4. Teensy unplugged from EBTKS. Hopefully not needed
- Run the Flash utility, you will probably need to press the white button on Teensy after selecting the .hex file (in the next step)
  - sudo ./teensy

(Cont'd)

- Browse to the EBTKS_Firmware_R23.hex - under operations pick program. Note that the rev number is not going to be R23. At this point you will need to press the white button on Teensy to enter programming mode.
- If you had to remove Teensy from EBTKS, reinstall it carefully, taking note of not just the outer 48 pins but also the two stips of 5 pins. The SD Card connector is close to the edge of the main PCB.

If you are reading this and you are a skilled Linux user, please take a little time to read the above update process for Windows to guide you, and then please, please, please write a similar version that is Linux specific, and identifies Linux related commands and things to check that helped you do the update. Send your text to philip for publication here (and the removal of this message). The rest of the Linux community will be very appreciative.

Mac

Waiting for someone to submit this section

Un-Bricking a Teensy 4.1

Usually the above firmware update process works, and the Teensy Loader automatically will erase and reprogram the Flash memory. The fall back is that the loader program will indicate that you need to press the little white button on the Teensy 4.1 module. When you do this, a LED near the micro-USB connector turns on at a low brightness red. Then the Teensy Loader erases and programs the Flash memory. During erase/program the red LED is bright. At the completion of programming the red LED turns off.

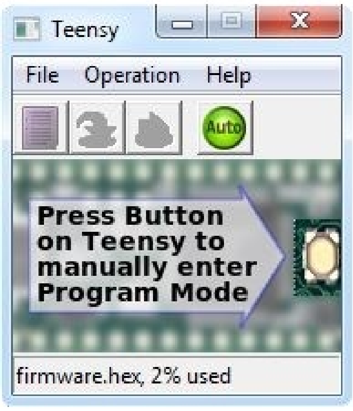
A common mistake I make is that I forget to disconnect any terminal emulation program (I use TeraTerm) from the serial-over-USB connection. So before proceeding in this section, make sure that you don’t have a serial connection to Teensy active, as that will block firmware updating.

Under some mostly undefined and rare situations the Teensy 4.1 can become non-responsive after a firmware update. This is usually associated with either a failure to complete the update process, or during firmware development when new bugs are being added to the code. The indication of this situation is the inability to update the firmware even when following the above instructions. When Teensy Loader indicates that you should press the little white button, the red led does not turn on or it does not remain on in the low brightness mode allowing progress to the erase and program steps. After repeated tries, you are unable to update or restore the firmware. A common term for this type of condition is that something is “bricked”. Teensy 4.1 has a backup backup plan.

When Teensy 4.1 is bricked, follow this procedure:

1. Turn the Series80 computer (HP85A/B, HP86A/B, HP87/87XM) power off or turn the power off and unplug EBTKS. You do not need to unplug the Teensy 4.1 module from EBTKS, and really, you shouldn’t because the pins are fragile, or the Teensy 4.1 is permanently soldered into EBTKS.
2. Connect the Teensy 4.1 to your PC/Mac/Linux system with the normal USB cable used for firmware updates. Start up the Teensy Loader

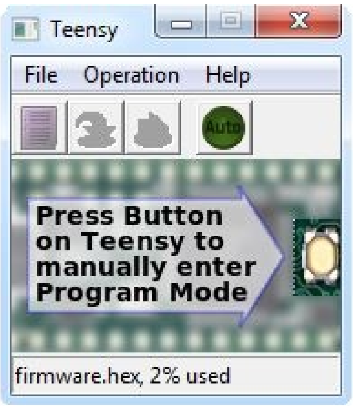
(Cont'd)



Press Button on Teensy to manually enter Program Mode

firmware.hex, 2% used

and turn off “Auto” mode (Auto icon is dark).

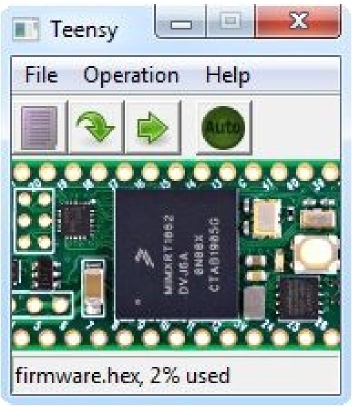


Press Button on Teensy to manually enter Program Mode

firmware.hex, 2% used

On the file menu, open HEX file, select the EBTKS firmware file. The rest of the Teensy Loader program window should show the message “Press Button on Teensy to manually enter Program Mode”

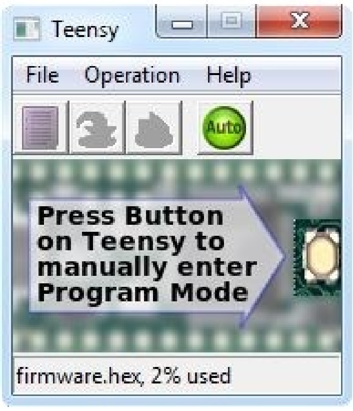
3. Press and hold the the white button on Teensy 4.1 for 15 seconds plus-or-minus 2 seconds. (i.e. from 13 to 17 seconds). At 13 seconds the red LED will flash at low brightness to let you know that 13 seconds has occurred. When this happens, release the button. Depending on the state of the Flash memory, there may be up to 10 seconds of the red LED being on at low brightness. This is followed by high brightness for 20 to 30 seconds. Finally, the red LED will turn off, and an orange LED near the white button will slowly flash. Teensy 4.1 has now been restored to its original factory settings.
4. Click the white button to initiate programming. The Teensy Loader will change to this view



firmware.hex, 2% used

(Cont'd)

and the red LED will be on at low brightness. You can now click the the down pointing green arrow which will update the firmware on Teensy 4.1 (during the programming the red LED will be bright). Finally click the green arrow that is pointing to the right to re-boot Teensy 4.1. To indicate that the firmware has been updated, the orange LED will give three very short blinks every 10 seconds. Click the dark auto icon to switch back to auto mode for the Teensy Loader



Press Button on Teensy to manually enter Program Mode

firmware.hex, 2% used

5. If you unplugged EBTKS, you can now plug it back into the Series80 computer.

Help

AUX ROM HELP “RULES OF THE ROAD”

You can type:

HELP

Shows the top level “85_Index” help file.

HELP THIS

Tries to find a help file called “THIS.txt” and if found, shows it. If not found, does a wildcard search for files that match “*THIS*”. If not found, generates “NOT FOUND” error. If exactly 1 is found, shows THAT file, otherwise if >1 is found, builds an “on the fly” help screen containing up to 64 lines with one link per line for each file found that matches “*THIS*” and shows that.

HELP THIS OR THAT

Same as “HELP THIS” except that it doesn’t try to find the exact file first, but rather immediately does a wildcard search for “*THIS*OR*THAT*”, then handles the result the same as “HELP THIS” with regards to 0, 1, or more results.

HELP *THIS

HELP *THIS*

HELP THIS*

HELP THIS OR THAT*

HELP THIS*OR*THAT

> When wildcards are included in the search term, then no searching is done for an exact match, and NO wildcards are ADDED to the search term, but rather a wildcard search is done using the exact wildcard search term provided. Results, again, are handled the same for 0, 1, or more results.

In creating Help files for use with the AUX ROM’s HELP function, some things are DITWAD (Do It This Way As Demanded) and other things are PROG (Please Remember Our Guidelines) . Few things are actually enforced, but if you follow DITWAD and PROG, everyone is likely to be happier.

DITWAD (Do It This Way As Demanded)

The Help screens are each stored in individual files of up to 64 lines of up to 32 chars each, with each line terminated by an end-of-line sequence (either LF or CR/LF). The HP-85 CRT alpha memory holds exactly that amount, hence the limit.

The file names for the Help screen files should be no more than 28 characters long (so that they will fit on a single line of the CRT along with the “<..>” tag header). This 28 character limit does NOT include the “.txt” file

(Cont'd)

extension, so with that extension, the filename can be a total of 32 characters long. The file name MUST end with a ".txt" file extension. The file names should not contain any of these characters:

SPACE COMMA / \ : $ @ ! * ?

The file contents must be plain, simple ASCII, as they will get displayed using the HP-85's font or character set, so don't try anything fancy or idiotic like a UTF-8 file or Unicode or blah-blah-blah.

When you create a Help screen file and name it, you must then place it in the appropriate sub-folder of the HELP85 sub-folder. If you're doing this on the EBTKS SD card, then the HELP85 folder will be in the root of the card ("/HELP85"), whereas on the Series 80 Emulator, the HELP85 folder will be wherever you've installed the Emulator (wherever the HP85.EXE file exists). In either case, the SDHOME$ function (on the real or emulated HP-85) will return the path to this 'home' directory, and THAT is where you'll find the HELP85 folder. Within that HELP85 folder are sub-folders 00, 01, 02, etc. You'll place your new Help file in one of those sub-folders, usually the highest numbered one. Each folder will hold approximately 100 help files before the next higher one should be created. NOTE: It's critical that these sub-folders be numbered sequentially and as 2-digit names, as that's what the AUX ROM expects and generates as it's searching for desired help files. Remember: don't use SPACE characters (or any of the above-mentioned verboten characters) in the name. Use an '_' (underscore) character in place of SPACE.

Within the limits of 64 lines of 32 characters, you can do whatever you want, just remember to keep to simple ASCII characters (look at the HP-85 character set from 32-127, removing the verboten characters, and those are the only ones you should use, although if you want to get tricky, you COULD use the 0-31 characters, except for CR and LF, although your editor might choke on some of those).

If the file is longer than 64 lines, all lines beyond 64 will be ignored. If lines are longer than 32 characters, the excess characters will wrap around to the next line and make a bloody mess out of everything, so don't do that. There is NO automatic "word breaking" if your lines are longer than 32 characters. The "wrap around" will happen exactly at 32 characters if you're foolish enough to have a line longer than 32.

LINKS in a Help screen begin with <..> and are immediately followed by the filename (minus the .txt suffix) of the file being linked to. For example, to put a link to "85_Index" in a Help screen, use this text:

```
<..>85_Index
```

The "<..>" tells the Help code that there's a link here and that the filename follows immediately after WITH NO SPACES! When loaded, the ".." will get replaced by a two digit number starting at "<00>" and increasing with each link up to a maximum of "<99>", so a maximum total of 100 links can be included on a single help page (if you can fit them in…) The link filename is terminated when a space is encountered or the end of the CRT line (the right edge of the CRT), whichever comes first. So, everything between "<..>" and the first space or right-side of the CRT line is the filename being linked to. There are no other limitations. The "<..>" can start at the left side or it can be in the middle of the screen, just so long as there's room for the entire filename to exist on the same line with the link tag.

At runtime, when the Help screen is being displayed, the "current LINK" is indicated by an underline beneath the "<xx>" for that link. You can move the "current link" marker with the cursor keys. If there are multiple link tags on a single line, up and down cursor pick the one that is CLOSEST to the same column. Left and right cursor only work when there ARE multiple links on the same (current) row.

If you have Help content that won't fit in a single 64-line file, you can make "sequential help files" by adding sequential two-digit numbers to the end of the file name (before the ".txt" extension). Then, when displayed by

(Cont'd)

HELP, the + and - keys can be used to move forward and backward (respectively) through the sequential files. NOTE: you may not use “00” as one of the numbers, as that is used internally in a way that will fail to link to a file with 00. An example would be:

```
HELP_TEST01.txt

HELP_TEST02.txt

HELP_TEST03.txt

etc
```

If you’re viewing HELP_TEST01 and the - key is pressed, nothing will happen, but if you press the + key, then HELP_TEST02 will be displayed. Another + key press will display HELP_TEST03, at which point further + key presses will do nothing, but - key presses can be used to move back to HELP_TEST02 and from there back to HELP_TEST01. If you need more than 99 sequential files, you’re writing a novel, not help screens, and should move to a more modern writing environment. It is expected that this will be a fairly rarely used feature, but my expectations have been dashed before.

PROG (Please Remember Our Guidelines)

This is where we come to the “what to do to make all of the Help screens look and work reasonably the same.”

If the filename specified with the HELP keyword isn’t found exactly, then it will search for filenames that ‘similar’. In order to facilitate this (that means, “to make this work for other people, you borking fidnip!”) it would be very good to use certain ‘keywords’ in your filenames, and in a certain order. It’s impossible to come up with a complete, exhaustive list of keywords to use, but below is a list of reasonably obvious ones. It’s important that everyone SPELLS THEM THE SAME, so that the search functions will work properly. For example, if different files use “IOROM”, “ROM_IO”, and “ROMIO”, only one of those are going to be found for the user, depending upon what the user types in. In NO case should you use “I/O ROM”, “IO ROM”, “ROMI/O”, “ROM I/O”, or “ROM IO”, because both ‘/’ and SPACE are verboten characters in file names. Further, if you’re making a Help screen on, say, the CONTROL keyword from the IO ROM, you would name the file “ROM_IO_CONTROL.txt” (upper/lower case doesn’t matter), not “CONTROL.txt” or “CONTROL_ROM_IO.txt”. You want to include all reasonable “search keywords” that will help the HELP statement find your file. If you do it right, then your file (along with others, potentially) will be found if the user types any of these commands:

```
HELP ROM IO

HELP CONTROL

HELP ROM CONTROL

HELP CONTROL ROM IO
```

Your file would still be found for all four of those, even if you named it “CONTROL_ROM.txt” or “CONTROL_ROM_IO.txt”, but it would be very nice, thank you, please, to have some semblance of consistency in the naming of these files.

So, when using multiple ‘keywords’ in your file name (think of them as “search terms”), they should be used in “highest-to-lowest” hierarchical order. You’re welcome to choose whichever keywords make the most sense (and fit within the 28 character limit), and make up your own, as well. But please keep the above in mind.

(Cont'd)

You do NOT need to use all ‘levels’ of keywords, and in general should OMIT levels of keywords unless they seem necessary for clarity OR for separating THIS help subject from foreseeable OTHER help subjects. For example, you will rarely want to place 83, 85, 86, 87 at the start of your Help, as any given collection of help screens will be targeted at EITHER the HP 85 family of computers OR targeted at the 86/87 family of computers, and their respective help files will be in different folder trees (HELP85 and HELP87). But if it’s REALLY important, then you would start off with one of those. Try to dredge up some logic, reason, and uncommon sense when choosing which levels of keywords to include, using everything that seems reasonable to use, and no more. Further, management reserves the right to rename your Help files without notice, consultation, or lip service. In general, avoid “model numbers” in filenames whenever possible (such as “82936A” … what the hell is THAT? Rather, use “ROM_DRAWER”; instead of “82937A”, use “HPIB”). Keep it simple, because HELP users need help, and if they have model numbers memorized, they probably don’t need help. You can put model numbers IN the Help screen itself, as THAT might be helpful…

Suggested keywords, in highest-to-lowest groupings, are:

DEVICE

- 83
- 85
- 85A
- 85B
- 86
- 86A
- 86B
- 87
- 87XM
- DISK (external)
- PRINTER (external)
- PLOTTER
- etc.

SUB-SYSTEM

- KEYBOARD
- CRT
- TAPE
- PRINTER (internal)
- DISK (86 “parallel interface” drives)
- etc.

IO MODULES

- HPIB
- GPIO
- SERIAL

(Cont'd)

- BCD
- PARALLEL
- MODEM
- RAM
- ROM
- EBTKS
- etc.

ROMS & BPGMS

- SYS (system ROMs)
- GRAPH
- PROGDEV
- MIKSAM
- LANG
- EXT
- ASM
- SYSEXT
- FORTH
- MATRIX
- IO
- EMS
- MS
- ED
- SERVICE
- AP
- PP
- AUX
- BPGM [name]

KEYWORDS

(actual keywords from Basic language provided by SYS/ROM/BPGM)

Ideally, there will be "hierarchical index pages," so that the top-level 'default' page (85_Index) will have links to all of the other "next level index pages", and those pages will have links to either further index pages or to actual files, so that a user could 'browse' or 'drill' their way down to the information they're looking for, as well as trying to search directly for it via a HELP xxxxx command.

With only one link per line, a single index could point to 64 files or other index pages. If we average that out to half of that, 32 links per index page… if 85_Index has 32 links to other index pages, and each of those averaged links to 32 files, just that would handle 1024 help files. So, using this scheme, the index is quite BROAD and SHALLOW, not more than 2 or 3 levels of indexing, most likely. It's conceivable that some Help pages will need to be linked to from more than one index, but that would definitely be an exception and not a rule.

Practice of Operation

Stuff goes here that goes deep into how EBTKS works.

For now, here is an interesting writeup from Everett Kaser (the author of the companion AUXROMs) on how it works.

Everett Kaser 10/27/2020 at 8:25 am

What Philip and Russell came up with *is* pretty incredible. Basically, you just have a *very* fast processor (a Teensy 4.1 microprocessor) hanging on the I/O bus (through level-shifting buffers), along with a few other minor electronics, and then *everything* else is software. The Teensy 4.1 (which has built-in 100 Mbit Ethernet, but not currently supported) runs at 600 Mhz, which compared to the Series 80's 613 Khz, is 963 cycles of the Teensy clock for every 1 cycle of the Series 80 clock. That gives Teensy a fair amount of time to do sh… uh… stuff during *every single cycle* of the Series 80 clocks.

Think of it as a cross between a logic analyzer and a software emulator. Teensy can watch the Series 80 bus for LMAs (Load Memory Address), compare the address to see if its something it's interested in, and then take actions (like setting up to read or write Teensy RAM that is being used to emulate Series 80 RAM, or providing ROM code bytes for emulated ROMs when the 85CPU is executing or just reading those ROMs, or doing 'silly' things like placing RAM right in the middle of ROM space (as it does for the AUXROMs, providing communications buffers between the Series 80 AUXROMs and the EBTKS Teensy processor, which is a little whacko when you think about it, because the Teensy processor is *emulating* the AUXROM, the RAM buried in it, as well as its own 'side' of the EBTKS/AUXROM communications that are being done through that emulated RAM 'window'). It can also watch *all* of the writes to CRT control registers and CRT RAM, so even though the HP-85 CRT's control registers aren't readable, you *can* read them via Teensy. Teensy can save and or write the *entire CRT RAM* contents in a couple of retrace cycles (quicker than the blink of an eye). Oh, and, of course, what started it all off, if you unplug the ribbon cables from the real tape drive, Teensy can "take over" the tape controller's I/O addresses and act just like the real tape controller, providing an emulated tape drive with emulated tape cartridges, much as my emulator does. Oh, and also I/O cards: HPIB with emulated Amigo disk drives. Oh, and…

whatever the hell you can think of.

Such as… Philip figured out that, using the 85CPU's HOLD line, the Teensy can actually do DMA to and from the 85's RAM, because while the 85 CPU is being held off, Teensy can drive the LMA, RD, WR lines itself allowing it to read or write whatever it wants as fast as the RD/WR lines can be toggled (i.e., as fast as the 85's four clocks run).

Speaking of which (HOLD), the 85 CPU *COULD* be held off forever, and Teensy *could* run an *emulator* of the 85 *much* faster than the original 85, so you'd just be using the 85's CRT and keyboard as, essentially, a dumb terminal, while the emulated 85 was running as fast as Teensy could run it, which would still work with the hardware I/O cards (of course, the speed would have to be down-tuned during those hardware conversations, but MOST of the time it's just executing ROM/RAM code, not doing I/O).

Or you could run an entirely different OS on the Teensy, turning the 85 into whatever computer you wanted.

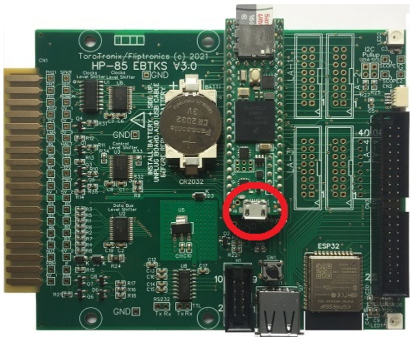
It's *all software*, and the levers of control are in your hands. The possibilities are, quite literally, endless. It just requires time and someone interested enough to do it. All the sources are, or will be, openly available, so it just requires someone to get interested, get their hands dirty, and "away we go…"

(You don't know how hard it's been not to talk about this stuff for the past 2-3 months! :-)

EBTKS Console

Console

EBTKS has a console interface that gives direct access to the internal operations of EBTKS via a serial-over-USB connection. At one end of this interface is the micro-USB connector on the Teensy 4.1 module, highlighted in this picture.



Connect a compatible USB cable from this port to to your desktop computer or laptop computer. The serial protocol is

```
ASCII character set
9600 Baud
8 data bits
No parity bit
1 stop bit

This is commonly just referred to as "9600 8N1" protocol
```

On your desktop or laptop computer you will need a terminal emulator program.
I like TeraTerm which is free.

Follow this link Setting up TeraTerm to see how to configure TeraTerm for communications with EBTKS.

Console Help Commands

These commands are all a single digit, and display a reminder menu of the available commands.

0	Show HELP page 0, list of other help pages
1	Show HELP page 1, Commands that display information
2	Show HELP page 2, Diagnostic Commands
3	Show HELP page 3, Directory listings, Date, Time
4	Show HELP page 4, Place Holder
5	Show HELP page 5, Commands for Developers
6	Show HELP page 6, Demo Commands

(Cont'd)

0

Show Help Page 0. This is a list of the other help pages

```
EBTKS> 0
EBTKS Control commands - not case-sensitive
0     Help for the help levels
1     Help for Display Information
2     Help for Diagnostic commands
3     Help for Directory and Time/Date Commands
5     Help for Developers
6     Help for Demo

The current Time and Date are displayed
```

1

Top

Show Help Page 1. Various commands that display information

```
EBTKS> 1
Commands to Display Information
Show log      Show the System Logfile
Show boot     Show the messages from the boot process, sent to Serial port
Show CRTboot  Show the messages sent to the CRT at startup
Show config   Show the CONFIG.TXT file
Show media    Show the Disk and Tape assignments
Show mb       Display current mailboxes and related data
Show CRTVis   Show what is visible on the CRT
Show CRTAll   Show all of the CRT ALPHA memory
Show key85_O  Display HP85 Special Keys in Octal
Show key85_D  Display HP85 Special Keys in Decimal
Show key87_O  Display HP87 Special Keys in Octal
Show key87_D  Display HP87 Special Keys in Decimal
Show other    Anything else is a file name path
```

2

Top

Show Help Page 2. Various Diagnostic commands

```
EBTKS> 2
Commands for Diagnostic
la setup      Set up the logic analyzer
la go         Start the logic analyzer
addr          Instantly show where HP85 is executing
kbdcode       Show key codes for next 10 characters in the keyboard buffer
clean log     Clean the Logfile on the SD Card
sdreadtimer   Test Reading with different start positions
SDCID         Display the CID information for the SD Card
PSRAMTest     Test the 8 MB PSRAM. You probably should do the PWO command when test has finished
ESP32 Prog    Activate a passthrough serial path to program the ESP32
pwo           Pulse PWO, resetting HP85 and EBTKS
```

(Cont'd)

3

Show Help Page 3. Directory and Time/Date Commands

```
EBTKS> 3
Directory and Date/Time Commands
dir tapes     Directory of available tapes
dir disks     Directory of available disks
dir roms      Directory of available ROMs
dir root      Directory of available ROMs
Date          Show current Date and Time (just typing 0 also works)
SetDate       Set the Date in MM/DD/YYYY format
SetTime       Set the Time in HH:MM 24 hour format
adj min       The U and D command will adjust minutes
adj hour      The U and D command will adjust hours
U             Increment the time by 1 minute or hour
D             Decrement the time by 1 minute or hour
```

4

Top

Show Help Page 4. Place Holder

```
EBTKS> 4
Commands for Auxiliary programs
```

5

Top

Show Help Page 5. Commands for Developers

```
EBTKS> 5
Commands for Developers (mostly Philip)
crt 1         Try and understand CRT Busy status timing
crt 2         Fast CRT Write Experiments
crt 3         Normal CRT Write Experiments
crt 4         Test screen Save and Restore
crt 5         Test writing text to HP86/87 CRT
```

6

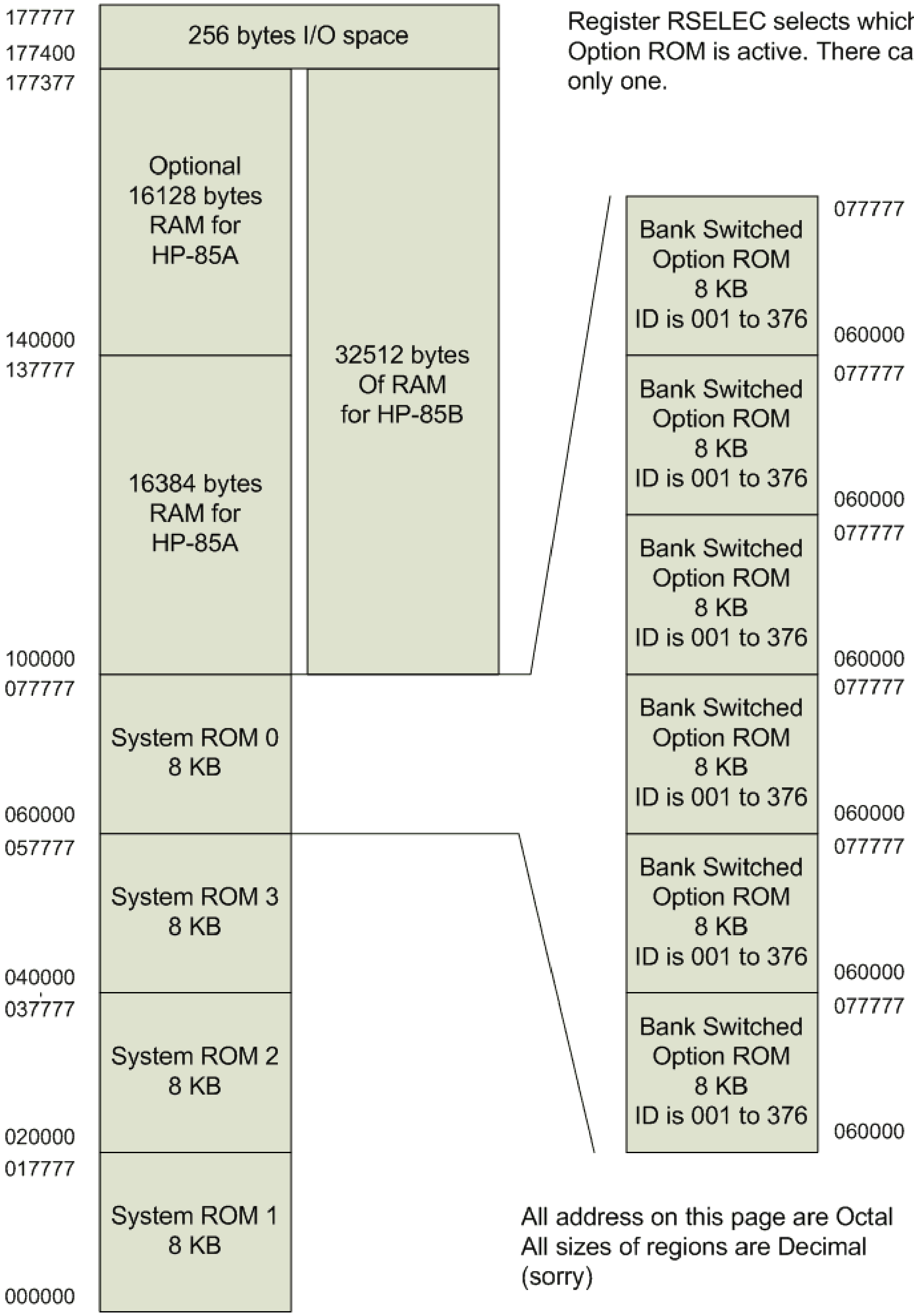
Top

Show Help Page 6. Demo Commands

```
EBTKS> 6
Commands for Demo
graphics test  Set graphics mode first
jay pi         Jay’s Pi calculator running on Teensy
```

EBTKS Memory Map for theAUXROM(s)

This is the standard memory layout for the HP-85 A and B

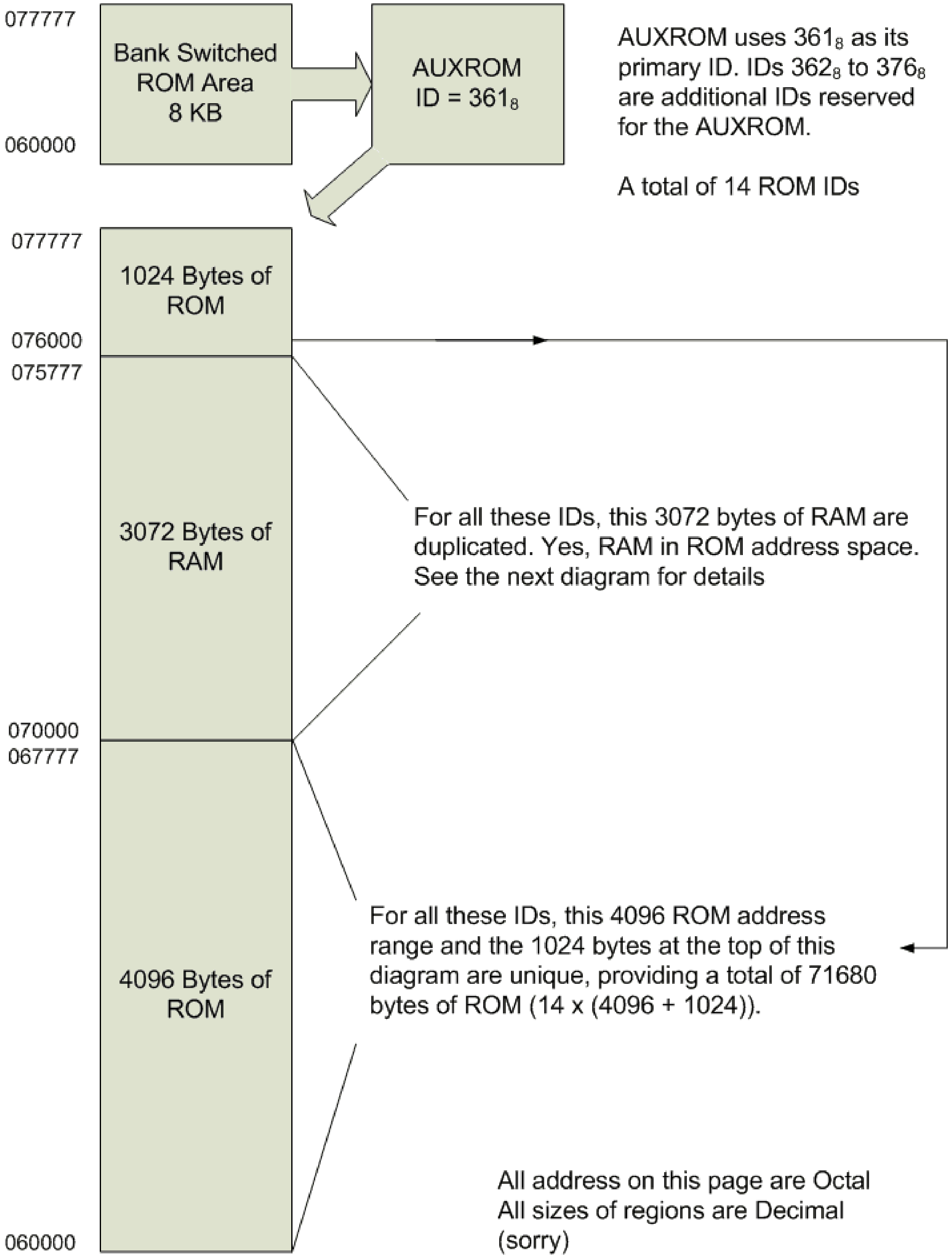


Register RSELEC selects which Option ROM is active. There can be only one.

All address on this page are Octal
All sizes of regions are Decimal
(sorry)

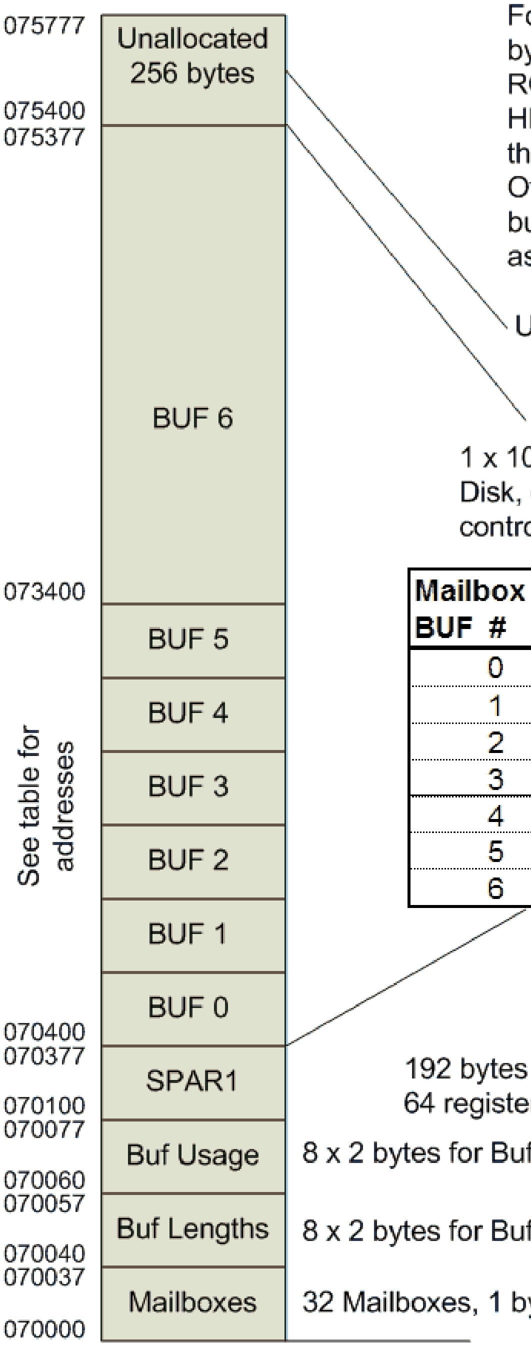
The AUXROM is implemented as one of the option ROMs on the right side of the above diagram

This is the detailed view of the AUXROM and its 13 additional extension ROMs


077777
060000
Bank Switched ROM Area 8 KB
AUXROM ID = 361_8
AUXROM uses 361_8 as its primary ID. IDs 362_8 to 376_8 are additional IDs reserved for the AUXROM.
A total of 14 ROM IDs
077777
076000
075777
070000
067777
060000
1024 Bytes of ROM
3072 Bytes of RAM
4096 Bytes of ROM
For all these IDs, this 3072 bytes of RAM are duplicated. Yes, RAM in ROM address space. See the next diagram for details
For all these IDs, this 4096 ROM address range and the 1024 bytes at the top of this diagram are unique, providing a total of 71680 bytes of ROM (14 x (4096 + 1024)).
All address on this page are Octal
All sizes of regions are Decimal
(sorry)

When any of the AUXROMs are selected, a shared/common block of RAM is available that can be both read and written by EBTKS and the software in the AUXROMs. To manage **write access** to this shared area, a set of 32 mailboxes are provided How Mailboxes and Buffers Work While the rest of each AUXROM is unique, there is a situation where duplication of code is advantageous Fast inter-ROM jumps .

Detailed view of the AUXROM sharedRAM area



For all AUXROM IDs (361_8 to 376_8), this 3072 bytes of RAM are duplicated. Yes, RAM in the ROM address space. EBTKS has full access. HP-85 has full access only when RSELEC is in the range 361_8 to 376_8

Of the 32 mailboxes, 7 are associated with the 7 buffers. The remaining 25 have not yet been assigned a function

Unallocated 256 bytes

1 x 1024 byte and 6 x 256 byte buffers for Tape, Disk, or other data transfers. Each Buffer N is controlled by a Mailbox N.

Mailbox & BUF #	Mailbox Addr	Buffer Start	Buffer End	Buffer Size
0	070000	070400	070777	256
1	070001	071000	071377	256
2	070002	071400	071777	256
3	070003	072000	072377	256
4	070004	072400	072777	256
5	070005	073000	073377	256
6	070006	073400	075377	1024

192 bytes reserved for SPAR1 Interrupt to save all 64 registers, SAD, and some spare

8 x 2 bytes for Buffer Usage Code (16 bytes total) See below for details

8 x 2 bytes for Buffer Lengths (16 bytes total) See below for details

32 Mailboxes, 1 byte each. See below for details

Regions are not to scale

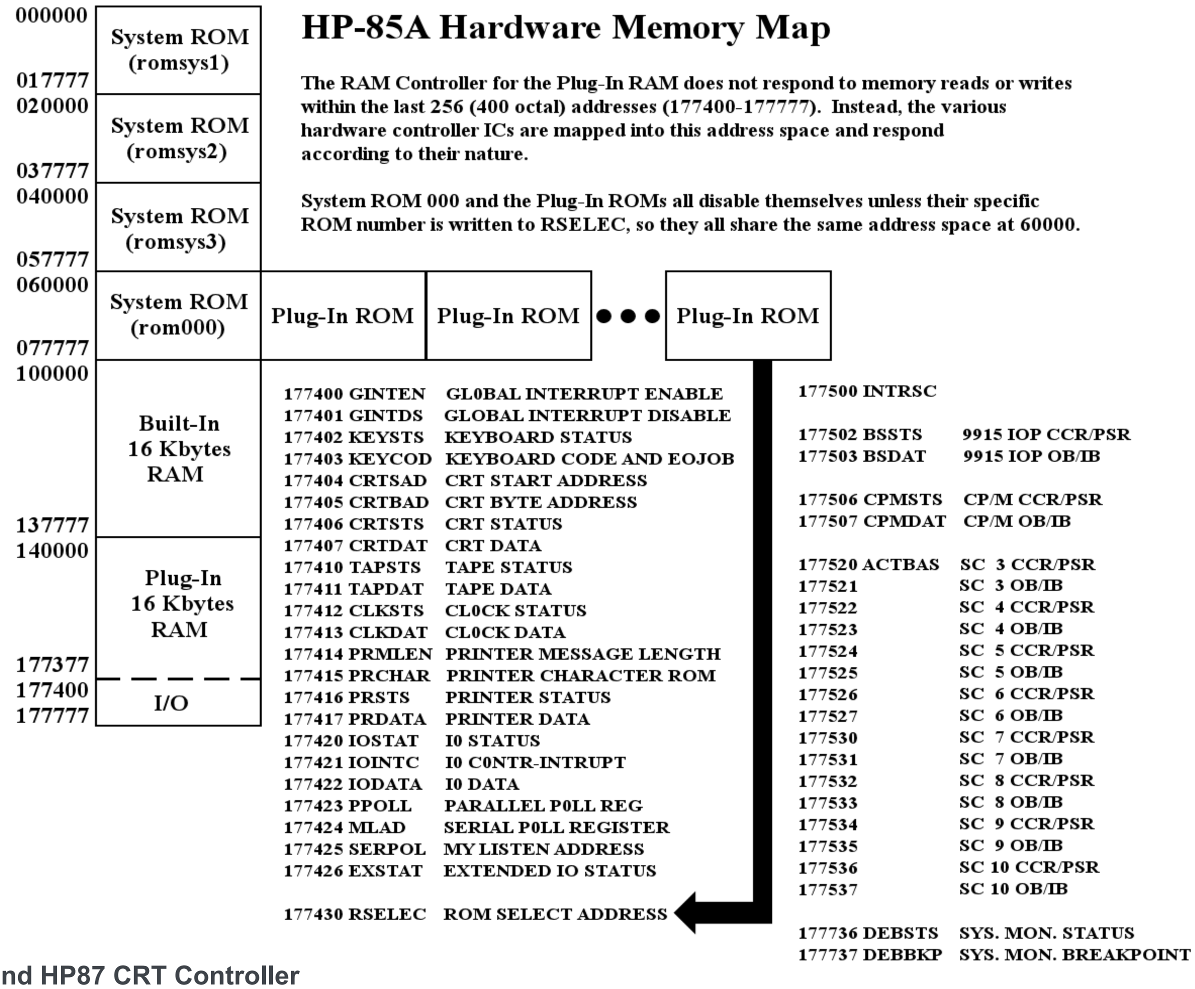
All address on this diagram are Octal
All sizes of regions are Decimal
(sorry)

AUXROM Shared RAM: Usage Plan

The details of how the shared RAM area is used is described in the paragraphs, starting here AUXROM Shared RAM: Buffers

EBTKS I/O Map

Here is another diagram taken from the HP-85 wiki authored by Everett Kaser that is on the Groups.io web site HP 85 System ROMs: Memory Layout (and Tokens and Allocation and...)



HP-85A Hardware Memory Map

The RAM Controller for the Plug-In RAM does not respond to memory reads or writes within the last 256 (400 octal) addresses (177400-177777). Instead, the various hardware controller ICs are mapped into this address space and respond according to their nature.

System ROM 000 and the Plug-In ROMs all disable themselves unless their specific ROM number is written to RSELEC, so they all share the same address space at 60000.

000000 017777	System ROM (romsys1)				
020000 037777	System ROM (romsys2)				
040000 057777	System ROM (romsys3)				
060000 077777	System ROM (rom000)	Plug-In ROM	Plug-In ROM	● ● ●	Plug-In ROM
100000 137777	Built-In 16 Kbytes RAM				
140000 177377	Plug-In 16 Kbytes RAM				
177400 177777	I/O				

177400 GINTEN	GL0BAL INTERRUPT ENABLE
177401 GINTDS	GLOBAL INTERRUPT DISABLE
177402 KEYSTS	KEYBOARD STATUS
177403 KEYCOD	KEYBOARD CODE AND EOJOB
177404 CRTSAD	CRT START ADDRESS
177405 CRTBAD	CRT BYTE ADDRESS
177406 CRTSTS	CRT STATUS
177407 CRTDAT	CRT DATA
177410 TAPSTS	TAPE STATUS
177411 TAPDAT	TAPE DATA
177412 CLKSTS	CL0CK STATUS
177413 CLKDAT	CL0CK DATA
177414 PRMLEN	PRINTER MESSAGE LENGTH
177415 PRCHAR	PRINTER CHARACTER ROM
177416 PRSTS	PRINTER STATUS
177417 PRDATA	PRINTER DATA
177420 IOSTAT	I0 STATUS
177421 IOINTC	I0 C0NTR-INTRUPT
177422 IODATA	I0 DATA
177423 PPOLL	PARALLEL P0LL REG
177424 MLAD	SERIAL P0LL REGISTER
177425 SERPOL	MY LISTEN ADDRESS
177426 EXSTAT	EXTENDED IO STATUS
177430 RSELEC	ROM SELECT ADDRESS

177500 INTRSC	
177502 BSSTS	9915 IOP CCR/PSR
177503 BSDAT	9915 IOP OB/IB
177506 CPMSTS	CP/M CCR/PSR
177507 CPMDAT	CP/M OB/IB
177520 ACTBAS	SC 3 CCR/PSR
177521	SC 3 OB/IB
177522	SC 4 CCR/PSR
177523	SC 4 OB/IB
177524	SC 5 CCR/PSR
177525	SC 5 OB/IB
177526	SC 6 CCR/PSR
177527	SC 6 OB/IB
177530	SC 7 CCR/PSR
177531	SC 7 OB/IB
177532	SC 8 CCR/PSR
177533	SC 8 OB/IB
177534	SC 9 CCR/PSR
177535	SC 9 OB/IB
177536	SC 10 CCR/PSR
177537	SC 10 OB/IB
177736 DEBSTS	SYS. MON. STATUS
177737 DEBBKP	SYS. MON. BREAKPOINT

HP86 and HP87 CRT Controller

I/O locations listed above from 177404 to 177407 are for the CRT controller used in the HP85A, HP85B and HP9915. This CRT controller is not used in the HP86 and HP87 computers, so these locations are unused. Instead, the HP 86 and HP87 have an alternative CRT controller (because the screen is 80x25 instead of 32x16) and the similar I/O registers are shown below.

	Full	I/O Offset	
	Address	Octal	Hex
HP86_87_CRTSAD	177700	300	0xC0
HP86_87_CRTBAD	177701	301	0xC1
HP86_87_CRTSTS	177702	302	0xC2
HP86_87_CRTDAT	177703	303	0xC3

HEYEBTKS

~~EBTKS reserves the following I/O addresses for communication between the AUXROM~~s and EBTKS. Writing a value to this mailbox I/O location tells EBTKS the function to be performed, and further information will be found in the shared memory .

	Full	I/O Offset	
	Address	Octal	Hex
HEYEBTKS	177740	340	0xE0

(Cont'd)

Extended Memory Registers

EBTKS can provide Extended memory through the implementation of an EMC (Extended Memory Controller) and up to 256 kB of Extended memory (at the time of writing this documentation. The 256 kB may change in the future). There are two types of EMC: Master and non-Master. There can only be one Master EMC in a system. The HP85B, and all versions of HP86 and HP87 have a built in EMC Master. The HP85A does not have the capability of supporting Extended memory, and does not have an EMC Master. The EMC implements 2 pointers which can be directly read and written on the EMC Master and are write only (for direct access) on non-Master EMCs, such as EBTKS. This means that the standard implementation of an EMC non-Master does not provide diagnostic access that EBTKS required during its design phase. To remedy this EBTKS includes an additional read-only register that allows accessing the direct content of PTR2 on EBTKS. See below for details.

For indirect access through PTR1 and PTR2, EBTKS responds to reads and writes as appropriate, if the value of the selected pointer is in the address range that EBTKS is configured to respond to. Configuration comes from the CONFIG.TXT file.

If enabled, the EMC emulation has 2 pointer registers accessed as follows

	Full	I/O Offset	
	Address	Octal	Hex
PTR1	177710	310	0xC8
PTR1-	177711	311	0xC9
PTR1+	177712	312	0xCA
PTR1-+	177713	313	0xCB
PTR2	177714	314	0xCC
PTR2-	177715	315	0xCD
PTR2+	177716	316	0xCE
PTR2-+	177717	317	0xCF

EMC Diagnostic PTR2

In HP85B, HP86 A or B, and HP87 A or XM, the Extended Memory Controller (EMC) master is on the main board and all other EMCs are in non-master mode, including EBTKS when it is implementing Extended Memory. When an EMC is in non-master mode, the 8 I/O locations are write only, and all EMCs are written to concurrently, and update their copies of the these two underlying Pointer Registers (PTR1 and PTR2) regardless of whether the pointer values indicate memory addresses within the range that the EMC is managing. When direct addressing mode read operations are performed on these locations, only the master responds with its copy. This created a problem for the design of EBTKS, when trying to debug its operation, as the above description shows there is no way to read back PTR1 and PTR2 to check that they are being updated and modified in synchronization with the other EMCs in the system. To solve this problem (and track down a very tricky bug) EBTKS has an I/O location that can be read to access the current internal value of PTR2. Although the issue for which it was created has now been resolved, this diagnostic read capability is still implemented, in case it may be needed in the future.

	Full	I/O Offset	
	Address	Octal	Hex
EBTKSPTR2	177760	360	0xF0

AUXROMs Internals

Make sure we cover these topics

- list of all the architectural details
- Calling conventions
- Mailboxes and Ownership
- Flowchart of transactions
- Impact of DMA on interrupts
- can’t do DMA to memory we provide (85A only)
- Performance measurements
- File sizes
- File name constraints
- Other gotchas

During Boot, the HP-85 only sees AUXROM ID 361 (octal). This image contains the correct signature in the bottom two bytes, the correct checksums in the top 4 bytes, and during initializing, it registers all the Keywords supported by all the AUXROMs. (A requirement of AUXROM 361 is that it is the gateway for all AUXROM functionality, and as such it must register all the Keywords. There is an upper limit of 254 Keywords.) The 13 additional AUXROM IDs are managed by AUXROM 361. These additional AUXROMs are not detected at boot time because they deliberately do not have the correct second byte of the ROM signature at address 060001. This allows for more ROM based software than the HP-85 was designed to support.

All 14 AUXROM IDs have the following:

- Unique ROM Regions for the lower 4096 bytes from addresses 060000 to 067777
- Unique ROM Regions for the upper 1024 bytes from addresses 076000 to 077777 (but see Fast inter-ROM jumps)
- A shared RAM area of 3072 bytes from address 070000 to 075777. This is nominally partitioned into six 256 byte blocks and one 1024 byte block. See this diagram. This partitioning is not enforced by the hardware, so alternative partitioning plans are possible, provided the firmware on EBTKS and AUXROM use the same plan

This RAM region (in what is normally considered ROM space) is visible to the HP-85 processor whenever any of the 14 AUXROM IDs are in the RSELEC register.

AUXROM Shared RAM: Buffers

There are 7 **Buffers** defined in the shared AUXROM RAM. As shown in the above diagram and table, 6 of the buffers are 256 bytes each, and there is 1 buffer of 1024 bytes. The buffers and their associated Length and Usage are numbered 0 to 6. These buffers are used to pass data between the HP-85 and EBTKS. Each buffer has an associated Mailbox (1 byte), Length (2 bytes), and Usage Code (2 bytes).

AUXROM Shared RAM: Mailboxes

There are 32 **Mailbox** locations, each 1 byte, that should only have the value 0 or 1. Mailboxes 0 to 6 are pre-assigned to manage access to Buffers 0 to 6. The remaining mailboxes (7 to 31) are available for as yet unwritten Keywords that may need to manage shared resources. I'll hold off the rest of the description of mailboxes until after the next few paragraphs.

AUXROM Shared RAM: Buffer Lengths

There are 8 **Buffer Lengths** that hold a 16 bit integer and are associated with their respective buffers (0 to 6). The 8th one is currently a spare. Total 16 bytes.

AUXROM Shared RAM: Usage Code

There are 8 **Usage Codes** that hold a 16 bit integer and are associated with their respective buffers (0 to 6). The 8th one is currently a spare. Total 16 bytes. Usage Codes are discussed in the AUXROM Keywords section. Briefly they are used to identify which keyword is to be processed, and on return, the success/failure of the processing the keyword.

How Mailboxes, Buffers, Buffer Lengths, and Usage Codes Work

The shared memory area contains 7 buffers (6 * 256 bytes and 1 * 1024). For each buffer there is

- A Buffer of either 256 or 1024 bytes
- A 1 byte Mailbox. Values 0 or 1
- A 2 byte Buffer Length. Values 0 to 256 or 0 to 1024
- A 2 byte Buffer Usage. Values 0 to N

Buffers are numbered 0 to 6, as are the Mailboxes, Buffer Lengths, and Buffer Usages. Each resource uses contiguous memory, i.e. all the mailboxes are contiguous. This diagram Detailed view of the AUXROM shared RAM area gives the addresses for each resource, with the lowest address for each resource being for Buffer 0, or its associated Mailbox, Length, or Usage Code.

The Mailboxes control the access rights for the Mailbox, Buffer, Length, and Usage

Each mailbox has a value of either 0 or 1, and the first 7 mailboxes have been assigned to manage the access to the 7 buffers that are located in the shared RAM and also the associated Length and Usage. When a mailbox has the value 0, the HP-85 AUXROM software may read or write to the associated buffer without interference from EBTKS. The same is true for the Length and Usage Code. While in this state, the EBTKS will not read or write the buffer. Likewise, when the mailbox has the value 1, the associated buffer can be read and written by EBTKS (and also the Length and Usage), but the HP-85 AUXROM software should not read or write the buffer.

(Cont'd)

Note that the ability to *read/write*, *read-only*, and *should not access* are not enforced by hardware. Rather, they are a convention that is managed by the current state of the mailboxes, and for reliable and predictable system operation, both the AUXROM software and the EBTKS firmware must abide by these rules.

While a buffers should not be read or written by the side that is not allowed to, it is expected that both sides can read the mailboxes at any time. This is the mechanism by which ownership of the buffers is passed back and forth between the EBTKS and the HP-85 AUXROM software.

Ownership can be given, but not taken

Since only the owner of mailbox (and the associated buffer, length and usage code) may write to a mailbox, only the owner may change its value. Thus, if the HP-85 has ownership (mailbox is 0), it can give ownership to EBTKS by writing a 1 to the mailbox, and concurrently the ownership of the buffer, length and usage code are also transferred. The correct way to do this is to make all required changes to the buffer, length, and usage code, and finally transfer ownership by changing the state of the mailbox. The same process occurs when EBTKS has ownership of a mailbox (and associated resources) and wants to pass ownership back to the HP-85. The idle state for ownership is that the AUXROMs own the resources.

All mailboxes are initialized to 0 when the system boots, giving ownership to the AUXROMs.

Mailbox State	AUXROM Access to Mailbox	AUXROM Access to Buffer, Length, Usage	EBTKS Access to Mailbox	EBTKS Access to Buffer, Length, Usage
0	Read and Write	Read and Write	Read Only	None
1	Read Only	None	Read and Write	Read and Write

All transfers of data between the HP-85 and EBTKS originate with some activity on the HP-85 side

Scenario 1: AUXROM wants to pass 256 bytes to EBTKS

For example, writing a FAT32 disk file to the SD card (This is not a totally complete example)

- Mailbox 0 is 0: Indicates HP-85 AUXROMs have ownership
- AUXROM software fills buffer 0 with 256 bytes of data, maybe as a result of BASIC function calls
- AUXROM software set the Buffer Length for buffer 0 to 256
- AUXROM software set the Buffer Usage for buffer 0 to a code indicating which operation is being performed (for example, write)
- When AUXROM software has finished writing data to the buffer, it sets Mailbox 0 to value 1. It has thus relinquished control of the Buffer 0, and its Length, Usage, and Mailbox, and has assigned them to EBTKS.
- AUXROM software then writes 0 (the number of the mailbox and buffer) to I/O register 177740, thus alerting EBTKS that something has changed
- EBTKS then processes the data that was sent. Depending on the task performed, if there is data to be returned, it is placed in the same buffer and the Length is set appropriately. The Usage is the standard way of passing back success or failure status for the function just performed. By convention, 0 indicates success, and other values may indicate success with additional info or information about a failing status.

(Cont'd)

This is not standardized, and different Keywords may use the returned Usage in their own specific way. After the buffer, Length and Usage have been updated, the final operation that EBTKS performs as part of the processing of the keyword is to pass ownership of the buffer, Length, and Usage back to AUXROMs by setting mailbox 0 back to value 0.

- While EBTKS was processing the data that the AUXROMs sent it, the scenario branches depending on whether there is more data to send
  - More data to send:

    The AUXROMs just loops, monitoring the mailbox waiting for it to return to 0, and when it does, it goes back to the beginning of this scenario to send more data. An example of this might be the AUXROMs sending 2KB of data, as multiple blocks of 256 bytes.
  - No more data to send:

    After handing off the buffer to the EBTKS, the AUXROMs can continue on to other tasks. Currently no keyword does this. They all wait for EBTKS to finish, since there is always the possibility of an error status being returned.

Scenario 2: A Keyword on the HP-85 is Executed to pass a string to EBTKS

For example, selecting a file name to be associated with the Tape Drive Emulation

- The HP-85 is executing a BASIC program and gets to a token for a keyword that is handled by AUXROMs. AUXROM software gets control. Note that by the time this token is reached, the address and length of the string will have been put on the operation stack that is managed by register R12.
- Mailbox 0 has value 0: Indicates HP-85 AUXROMs have ownership
- Code in the AUXROM performs any necessary tasks prior to setting up a buffer with a copy of the string.
- AUXROM software copies the string into buffer 0
- AUXROM software set the buffer Length for buffer 0 to the length of the string
- AUXROM software set the Buffer Usage for buffer 0 to a code indicating that the statement to be processed is “set filename of emulated tape”
- AUXROM software has now finished setting things up, and can now let EBTKS take over. AUXROM changes Mailbox 0 to value 1. It has thus relinquished control of the buffer 0, and its Length, Usage, and Mailbox, are assigned to EBTKS.
- AUXROM software then writes 0 (the number of the mailbox and buffer) to I/O register 177740, thus alerting EBTKS that something has changed
- AUXROM enters a loop monitoring Mailbox 0, waiting for it to change back to 0
- EBTKS sees the write to 177740 and knows there is a new task to be performed, and the value written is 0, so it knows that the information about what is to be done is in buffer 0 (and its associated Length and Usage Code)
- EBTKS reads the Usage code, sees that it is a File name setting for the emulated Tape Drive. It expects that the file name is in buffer 0 and the Length is in the associated Length location.
- EBTKS fetches the string and does whatever is needed to treat it as a directory path and file name, and associate that file with the Tape Drive emulation.
- Depending on success or failure, EBTKS sets the Usage value either to 0 for success, or some other value to indicate what type of error occurred

(Cont'd)

- Finally, the EBTKS writes a 0 to the mailbox. This ends its involvement with the BASIC Statement.
- The AUXROM software (which has been looping, watching the mailbox) sees that ownership has been returned. It handles the value in the Usage location, that indicates either success or failure.
- The AUXROM software does any cleanup, and exits back to wherever it was called from.

Scenario 3: AUXROM wants to receive 256 bytes from EBTKS

For example, getting data from an ADC (This is not a totally complete example)

- A set of command bytes (maybe looking like an HPIB request to an ADC) is placed in Buffer 3, possible info may include voltage range, ADC channel, sample rate, data format, number of samples
- The Length value for buffer 3 is set appropriately
- The Usage value for buffer 3 is set appropriately
- Mailbox 3 is set to 1, thus passing ownership to EBTKS
- AUXROM software then writes 3 (the number of the mailbox and buffer) to I/O register 177740, thus alerting EBTKS that something has changed
- AUXROM enters a loop monitoring Mailbox 3, waiting for it to change back to 0
- EBTKS performs the requested ADC operation, and places the results in Buffer 3. It also sets the Length and Usage values.
- Lastly, EBTKS sets Mailbox 3 back to 0, relinquishing ownership.
- If more data is to be received, a different buffer could be passed to EBTKS, to continue to receive data, while the buffer just received (3) could be processed.
- Using 2 buffers this way (1 being processed, 1 receiving) the system ping-pongs between the two buffers, maximizing data transfer rate and system utilization.

Fast inter-ROM jumps

It is currently expected that there will be a small block of code in the upper AUXROM region that is identical in all of the AUXROMs, and this will support very fast switching between different AUXROMs by changing the RSELEC register and doing a JSB to a branch table in the destination AUXROM. This will be much faster than how traditional HP-85 ROMS branch between co-operating ROMs (like EMS and EDISK ROMs)

Logic Analyzer

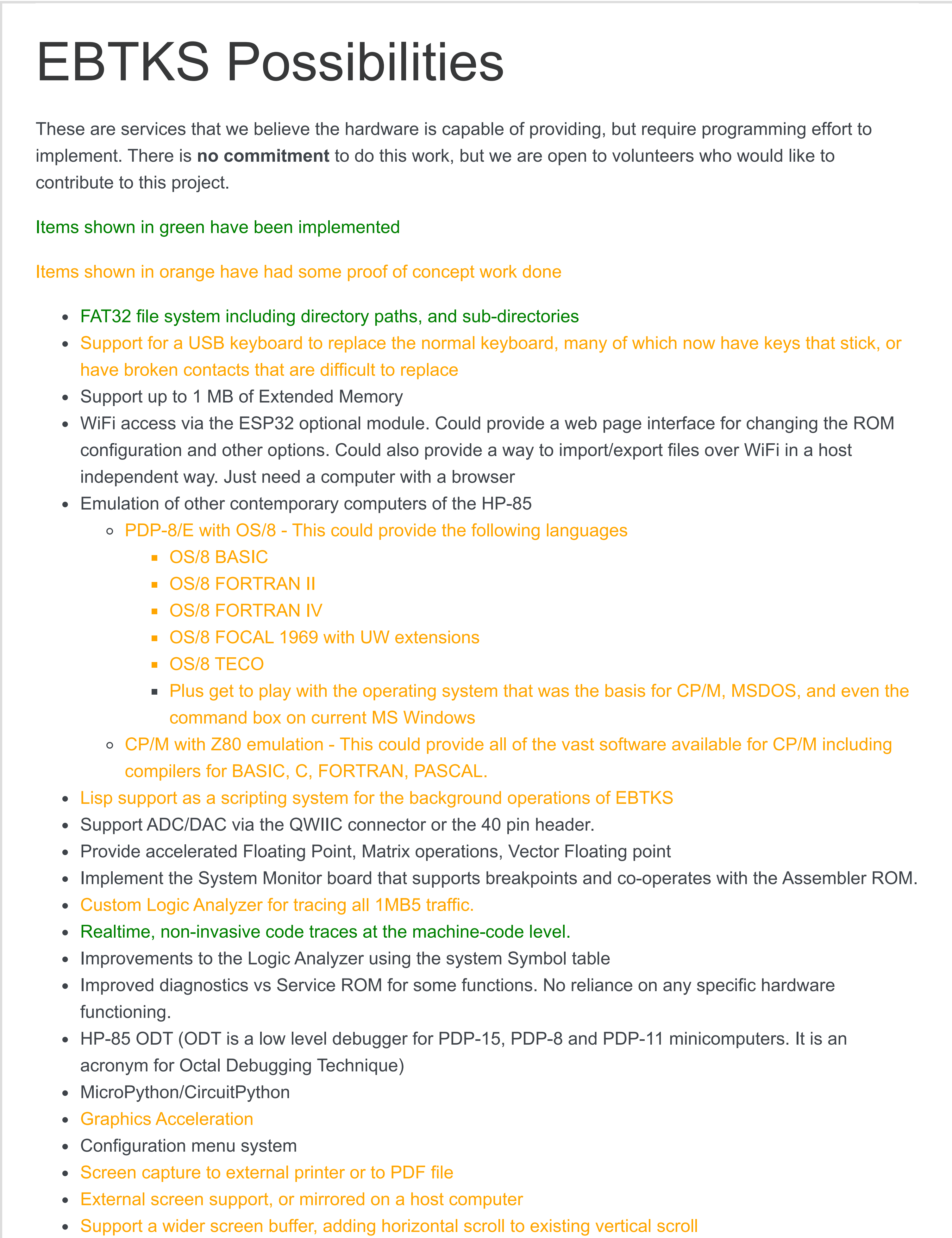
EBTKS implements a real-time non-invasive logic analyzer that can trace all bus activity.

Need to document how to set it up.

Need to document how to select trigger conditions, and related strategies

Need to document how to interpret the results

Need to get back into the code and add tracing of DMA transactions



EBTKS Possibilities

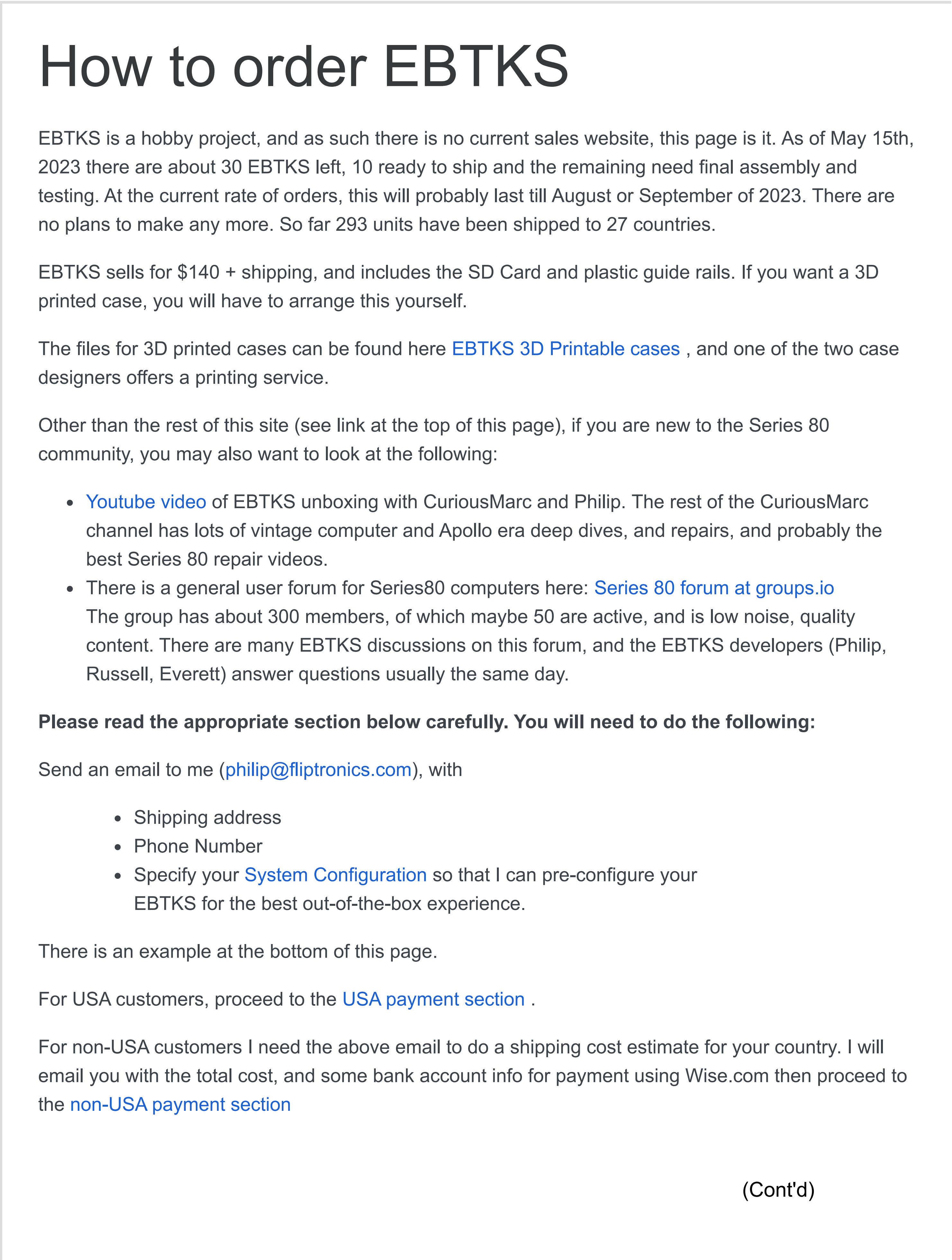
These are services that we believe the hardware is capable of providing, but require programming effort to implement. There is **no commitment** to do this work, but we are open to volunteers who would like to contribute to this project.

Items shown in green have been implemented

Items shown in orange have had some proof of concept work done

- FAT32 file system including directory paths, and sub-directories
- Support for a USB keyboard to replace the normal keyboard, many of which now have keys that stick, or have broken contacts that are difficult to replace
- Support up to 1 MB of Extended Memory
- WiFi access via the ESP32 optional module. Could provide a web page interface for changing the ROM configuration and other options. Could also provide a way to import/export files over WiFi in a host independent way. Just need a computer with a browser
- Emulation of other contemporary computers of the HP-85
    - PDP-8/E with OS/8 - This could provide the following languages
        - OS/8 BASIC
        - OS/8 FORTRAN II
        - OS/8 FORTRAN IV
        - OS/8 FOCAL 1969 with UW extensions
        - OS/8 TECO
        - Plus get to play with the operating system that was the basis for CP/M, MSDOS, and even the command box on current MS Windows
    - CP/M with Z80 emulation - This could provide all of the vast software available for CP/M including compilers for BASIC, C, FORTRAN, PASCAL.
- Lisp support as a scripting system for the background operations of EBTKS
- Support ADC/DAC via the QWIIC connector or the 40 pin header.
- Provide accelerated Floating Point, Matrix operations, Vector Floating point
- Implement the System Monitor board that supports breakpoints and co-operates with the Assembler ROM.
- Custom Logic Analyzer for tracing all 1MB5 traffic.
- Realtime, non-invasive code traces at the machine-code level.
- Improvements to the Logic Analyzer using the system Symbol table
- Improved diagnostics vs Service ROM for some functions. No reliance on any specific hardware functioning.
- HP-85 ODT (ODT is a low level debugger for PDP-15, PDP-8 and PDP-11 minicomputers. It is an acronym for Octal Debugging Technique)
- MicroPython/CircuitPython
- Graphics Acceleration
- Configuration menu system
- Screen capture to external printer or to PDF file
- External screen support, or mirrored on a host computer
- Support a wider screen buffer, adding horizontal scroll to existing vertical scroll

MORE Hardware and Software info needed here



How to order EBTKS

EBTKS is a hobby project, and as such there is no current sales website, this page is it. As of May 15th, 2023 there are about 30 EBTKS left, 10 ready to ship and the remaining need final assembly and testing. At the current rate of orders, this will probably last till August or September of 2023. There are no plans to make any more. So far 293 units have been shipped to 27 countries.

EBTKS sells for $140 + shipping, and includes the SD Card and plastic guide rails. If you want a 3D printed case, you will have to arrange this yourself.

The files for 3D printed cases can be found here EBTKS 3D Printable cases , and one of the two case designers offers a printing service.

Other than the rest of this site (see link at the top of this page), if you are new to the Series 80 community, you may also want to look at the following:

- Youtube video of EBTKS unboxing with CuriousMarc and Philip. The rest of the CuriousMarc channel has lots of vintage computer and Apollo era deep dives, and repairs, and probably the best Series 80 repair videos.
- There is a general user forum for Series80 computers here: Series 80 forum at groups.io The group has about 300 members, of which maybe 50 are active, and is low noise, quality content. There are many EBTKS discussions on this forum, and the EBTKS developers (Philip, Russell, Everett) answer questions usually the same day.

Please read the appropriate section below carefully. You will need to do the following:

Send an email to me (philip@fliptronics.com), with

- Shipping address
- Phone Number
- Specify your System Configuration so that I can pre-configure your EBTKS for the best out-of-the-box experience.

There is an example at the bottom of this page.

For USA customers, proceed to the USA payment section .

For non-USA customers I need the above email to do a shipping cost estimate for your country. I will email you with the total cost, and some bank account info for payment using Wise.com then proceed to the non-USA payment section

(Cont'd)

Shipping and Payment for **USAcustomers**

USA Shipping is $10 for any number of EBTKS. Shipment is via US post with tracking.

Payment - USA

I use Zelle for payment transfer, which is direct bank to bank transfer, without any fees in most cases. This does not support credit cards, it requires you to have money in your bank account.

You should be able to go to your online banking and find an option to make a payment with Zelle (most US banks support this).

My Zelle account is at Bank of America. You will need the following information:

- First Name: Philip
- Last Name: Freidin
- email: philip@fliptronics.com

Please make your payment (covering any fees if needed) such that I receive $140 per EBTKS + $10 for shipping.

A note about Zelle:

> - If you understand how scams work, you can just skip this
> - This is just my public service announcement
> - To my knowledge, no EBTKS buyer has been scammed via Zelle
> - I provide this info, because many EBTKS buyers have never used Zelle.
>
> I use Zelle for payment since most US banks support it, it is free to use, has no fees, and the transfers happen the same day. While Zelle is not a particularly new service, it has become a new target for scams, and has been in the news. Using Zelle to pay me does not make you more or less likely to be exposed to a scam. Invariably scams involving payment services (like Zelle, Western Union and PayPal) require you to be tricked into sending money to the scammer. This includes phone calls that claim to be from someone at your bank asking you to log into your account, and then doing what they tell you. Often associated with some fake issue that is urgent. Sometimes with phone caller ID that indicates it is from your bank, or IRS, or FBI, or other government department. Caller ID can be faked, so never trust it when money is involved. Anyone asking you to pay for something with a gift card that you need to buy, is a scam. Any government department wanting you to pay with a gift card is an insane scam, the government does not need gift cards. Threats from “government departments” that require you to pay right now, or you will be arrested, is a scam. The government does not phone you with such threats, it’s a scam. There are many

(Cont'd)

other indicators, but these are all ones I have experienced (and never been scammed because I know they are fake). As a general rule, if you get a call from someone claiming to be from your bank or government department i.e. IRS, and they want any of your login information, or want you to log into your account, it is probably a scam! Even if caller ID indicates it is from your bank etc., this is unreliable and can be faked.

What should you do

Ask for specifics of the claimed issue and the account number involved. If they can't provide that info, just end the call. If it isn't a scam, keep that info, end the phone call, and call your bank using the phone number provided on your bank statements. Your initiating the call using a legitimate bank phone number can get you (via the normal phone menu and queues) to talk to a real bank representative, and find out if there is a real issue.

You can google for “Zelle scams” to read examples.

Here is a sad example

Shipping and Payment for **non-USA customers**

Please send the email as specified above. Currently I have to figure out shipping costs on a case by case basis, so I need your full address and phone number. I will reply to your email within 2 days with shipping information and cost.
Unfortunately, for some countries there is no low cost shipping option.

Payment non-USA

I use Wise.com for non-USA customers.

Although not necessary, if you are creating a new Wise account, you can create your account by using my invite link, which may give me some extra money in the future (if you ever do a transfer over $300) and give you a discount (a few dollars) on your first transfer, paying for EBTKS.

In my email reply to you that provides shipping cost, I will also provide the information for step (f) below.

Invite link: Wise.com invite link

=== Payment with Wise.com

a. If you don't already have an account on Wise, create an account
b. Go to the account home
c. Press the “Send Money” button
d. In the upper currency box, select your currency, and in the lower currency box, select USD

(Cont'd)

e. In the “Recipient gets” field next to the USD, enter the amount in US Dollars, and it will calculate the amount in your currency in the upper box “You send exactly”

f. Fill in the Bank details on the page labeled “Send to someone else” using the information I supplied in my email providing shipping cost and my bank account details.

Address and System Configuration

This section is for all customers

I will pre-configure your EBTKS to give the best Out-of-the-Box experience. Mostly this means editing the CONFIG.TXT file. All of the settings that I will do for you can be changed easily with a text editor.

After you have made your payment, please email me with the following information (don’t include the quote (”) characters)

A. Your full name

B. Your Phone Number

C. Your shipping address.

- Street Name and Number (or Number and Name)
  - (however addresses are normally written in your country)
- Apartment/Suite number/Post box number
- City/District/County/Municipality/Hamlet/…
  - (however addresses are normally written in your country)
- State (if appropriate)
- Post Code or Zip Code
- Country
  - (and anything else to help get the package to you)

D. The computer you will be initially plugging EBTKS into, One of:

```
HP83        HP9915A     HP85A       HP85B       HP9915B
HP86A       HP86B       HP87        HP87XM
```

E. If it is a model that has a tape drive (HP85A/B, 9915A/B)

- enter either
  - “Tape Drive Disabled”   or
  - “Tape Drive Enabled”
- (disabled means either by physical removal, or by unplugging the two flat flexes cables.)
- For all other models, enter “N/A” (not applicable)

F. RAM for HP85A,83, 9915A

(Cont'd)

    - Enter “EBTKS provides 16 kB RAM” (if you have a 16 kB RAM module you will need to unplug it)
    - For all others Series 80 models, enter “N/A”

G. EMC memory for HP85B, 9915B, HP86A/86B, HP87/87XM

    - Model (same as C above) + how many 82908A (64 kB) and 82909A (128 kB) will be plugged in and how much EMC memory do you want EBTKS to provide, in banks of 32 kB each. 0 to 8
    - For non-EMC models (83, 85A, 9915A): “N/A”

For example:

```
HP85B + 64 kB + 64 kB + EBTKS 8 banks        if you have 2 x 82908A
HP86A + 128 kB + EBTKS 2 banks               if you have 1 x 82909A
HP87XM + 128 kB + 64 kB + EBTKS 0 banks      if you have a 82909A and a 82908A
HP87  + EBTKS 4 banks                        if you don't have memory modules
```

Note: On HP86/87 the boot up time is impacted by how much EMC memory there is. Working with CONFIG.TXT

H. Make sure if you have any I/O modules plugged in (like HPIB) that the select code is not 3, which is used by EBTKS. After you have checked you computer, enter “No other modules use Select Code 3”

For all Series80 models, I will assume that you have unplugged any ROM drawer. EBTKS can be configured for any ROM combination. I will assume the default built-in ROMS for HP85B, 9915B, and all 86 and 87 models. HPIB and other modules can be plugged in, but must not use select code 3.

Here is an example of a correctly formatted email:

```
A)  Mark Smith
B)  (123) 555-1212
C)  1234 HP85 Way
    Apt 488
    Springfield  TX 23456
    USA
D)  HP85B
E)  Tape Drive disabled
F)  N/A
G)  HP85B + 64 kB + 64 kB + EBTKS 8 banks
H)  No other modules use Select Code 3
```

When sending me this email (philip@fliptronics.com) , you are welcome to ask any questions or clarifications.

Application Notes

Using EBTKS in multiple Series80 Computers

Lots of wise text explaining multiple SD Cards. Need for different base file set for 85A/B and 86/87.

SD Card contents differences between 85A/B and 86/87¶

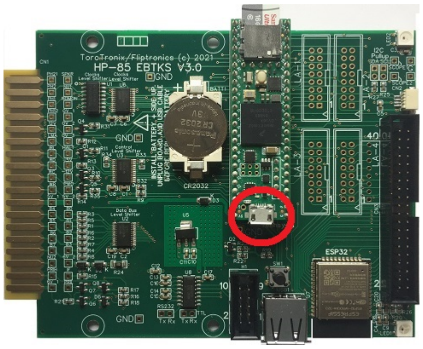
Lots of wise text explaining differences in the default floppy, and the /TOK directory and also the default media assignments

Editing CONFIG.TXT with BASIC programs

Lots of wise text explaining about need to reboot, and need to be very, very, careful not to break JSON formatting

Setting up TeraTerm

TeraTerm is a free terminal emulator program that can run on a PC and connect via a USB cable to the Teensy 4.1 module on EBTKS. The usb port on the Teensy 4.1 is called the console port.



Connecting EBTKS Teensy Console port to your computer

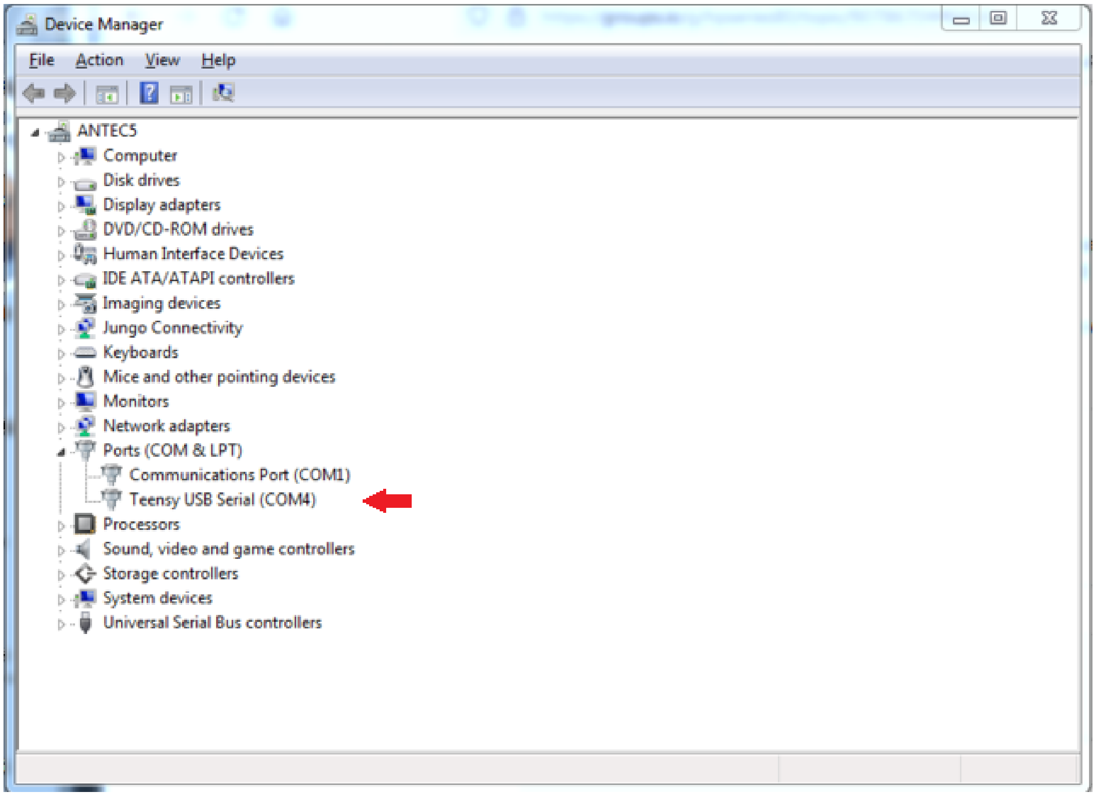
To use the console, you need to connect a USB micro-B cable to the non SD Card end of the Teensy 4.1 module to your PC/MAC, and run a terminal emulator program. My preference is TeraTerm.

(Cont'd)

The Console commands are documented here EBTKS Console

When you connect the USB cable to the PC (assuming EBTKS is powered by being in a powered Series 80 computer) Windows will go to the Internet and find the device driver for the Teensy 4.1 virtual COM port, and then install it.

You can check that it was successful by using the device manager. You should see some thing like this (showing COM4 was assigned):



```
ANTEC5
  Computer
  Disk drives
  Display adapters
  DVD/CD-ROM drives
  Human Interface Devices
  IDE ATA/ATAPI controllers
  Imaging devices
  Jungo Connectivity
  Keyboards
  Mice and other pointing devices
  Monitors
  Network adapters
  Ports (COM & LPT)
    Communications Port (COM1)
    Teensy USB Serial (COM4)
  Processors
  Sound, video and game controllers
  Storage controllers
  System devices
  Universal Serial Bus controllers
```

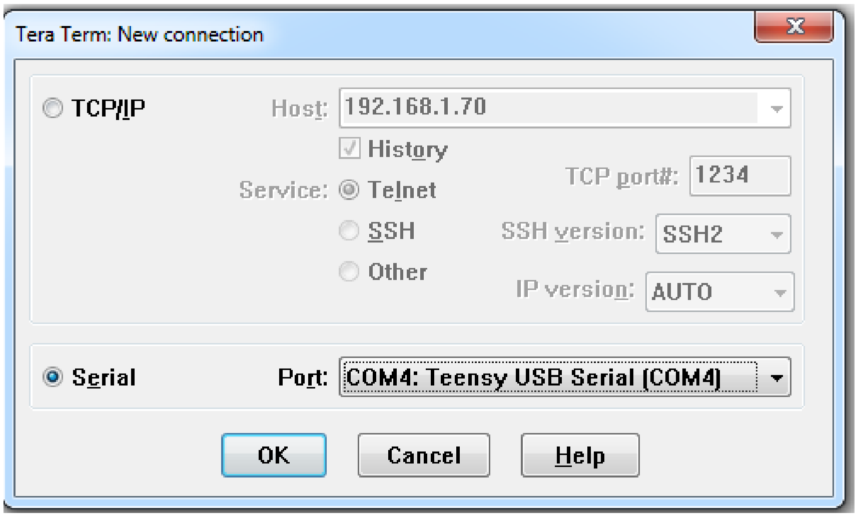
Setting up TeraTerm for EBTKS

EBTKS also has a set of built in diagnostic tool (primarily for Philip’s use) that are minimally listed and poorly documented here: EBTKS Console

These commands are available over a virtual serial link over USB. There is a micro USB connector on the Teensy module at the other end from the SD Card. You will need a terminal emulator program on your PC/MAC that connects to the COM port that will become available after the connection is made. The protocol is vanilla 9600 Baud 8N1. I use TeraTerm, which is a free terminal emulator.

After installing TeraTerm, on the menu line, select File->New Connection and select Serial and the virtual COM port that Teensy provides over the USB connection, as shown in thabove picture. This will require that Teensy is plugged into your Series 80 computer, and power is on.

(Cont'd)



Tera Term: New connection

○ TCP/IP Host: 192.168.1.70

☑ History

Service: ● Telnet TCP port#: 1234

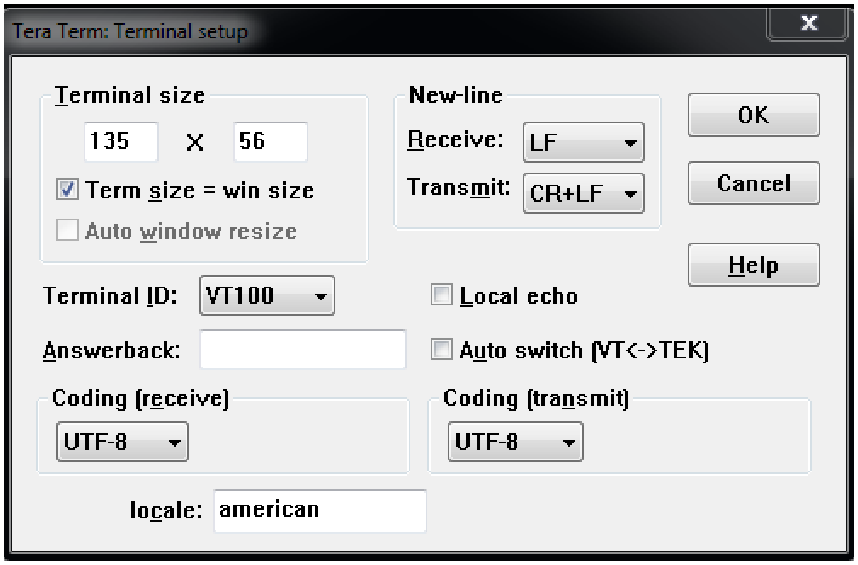
○ SSH SSH version: SSH2

○ Other IP version: AUTO

● Serial Port: COM4: Teensy USB Serial (COM4)

Then, continue with configuring this connection:

On the Setup -> Terminal page, do this:



Tera Term: Terminal setup

Terminal size

135 X 56

☑ Term size = win size

☐ Auto window resize

New-line

Receive: LF

Transmit: CR+LF

Terminal ID: VT100

Answerback:

☐ Local echo

☐ Auto switch (VT<->TEK)

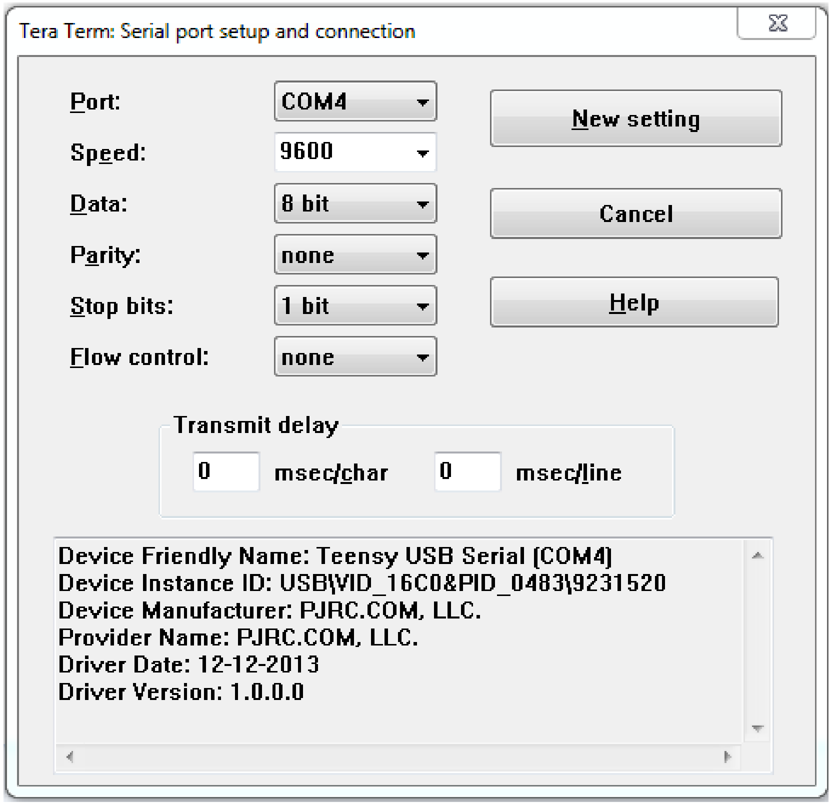
Coding (receive): UTF-8

Coding (transmit): UTF-8

locale: american

(Cont'd)

On the Setup -> Serial Port page, do this:



Tera Term: Serial port setup and connection

Port: COM4
Speed: 9600
Data: 8 bit
Parity: none
Stop bits: 1 bit
Flow control: none

Transmit delay
0 msec/char 0 msec/line

```
Device Friendly Name: Teensy USB Serial (COM4)
Device Instance ID: USB\VID_16C0&PID_0483\9231520
Device Manufacturer: PJRC.COM, LLC.
Provider Name: PJRC.COM, LLC.
Driver Date: 12-12-2013
Driver Version: 1.0.0.0
```

With your Series 80 computer + EBTKS running, you should get an EBTKS console prompt like this:

EBTKS>

The Console commands are minimally documented here EBTKS Console

Setting Time and Date

Currently there is no way to adjust the Time or Date from the Series 80 computer, but it can be done with the EBTKS Console.

Follow this link Setting up TeraTerm to see how to configure TeraTerm for communications with EBTKS, or as a guide to configure some other terminal emulator.

Once you have your terminal emulator set up, and connected to the EBTKS, you should get a prompt for a command:

```
EBTKS>
```

Setting the Date

To change the date, the setdate command requires the full month/day/year as shown in these examples:

```
EBTKS>setdate   MM/DD/YYYY
```

```
EBTKS>setdate   03/19/2023
```

(Cont'd)

Setting the Time

To change the time, the settime command requires both hours and minutes as shown in these examples:

EBTKS>settime   HH:MM

EBTKS>settime   14:35   (2:35 pm)

If you want to get the exact second, set the minutes part to the next minute, and type the enter key when the new minute starts.

Adjusting the Date and Time

Sometimes it is much easier to adjust the time or date rather than entering it as above. To facilitate this, EBTKS has two adjustment modes:

- adj hours (the initial default mode)
- adj min

In both modes, the "U" and "D" characters change either the hours or minutes up/down by 1, and then display the new date or time. After typing U/D you still need to type the enter key.

```
EBTKS> 0

EBTKS Control commands - not case-sensitive

0     Help for the help levels
1     Help for Display Information
2     Help for Diagnostic commands
3     Help for Directory and Time/Date Commands
5     Help for Developers
6     Help for Demo
8     Simple Logic Analyzer for emulated 1MB5

The time is 19:03:28 on Sunday the 19th of March 2023

EBTKS> u
Now 20:03
EBTKS> u
Now 21:03
EBTKS> d
Now 20:04
EBTKS> d
Now 19:04
EBTKS> adj min
Up/Down adjust (U/D) is now minutes
EBTKS> d
Now 19:03
EBTKS> d
Now 19:02
EBTKS> d
Now 19:01
EBTKS> u
Now 19:02
EBTKS> u
```

(Cont'd)

```
Now 19:03
EBTKS> u
Now 19:04
EBTKS> adj hour
Up/Down adjust (U/D) is now hours
EBTKS> 0

EBTKS Control commands - not case-sensitive

0     Help for the help levels
1     Help for Display Information
2     Help for Diagnostic commands
3     Help for Directory and Time/Date Commands
5     Help for Developers
6     Help for Demo
8     Simple Logic Analyzer for emulated 1MB5

The time is 19:04:42 on Sunday the 19th of March 2023

EBTKS>
```

More on MOUNT

EBTKS implements disks and tapes as files that are stored in the /disks/ and /tapes/ directories, respectively. An initial set of associations is provided in CONFIG.TXT .

You can change these “on-the-fly” using the MOUNT command. The associations between a msus$ and the files that represents disks or tapes are lost when the Series 80 computer is turned off, and when turned back on, the associations in CONFIG.TXT are restored.

If you want to make the association permanent, then the CONFIG.TXT file needs to be edited. This is documented here Disk Drives . This description is quite detailed, as the complete mangemnt of storage is fairly complex. If you aren’t trying to do anything exotic, the easiest shortcut is to just change the filename in an existing msus$ association. This is described below.

A virtual disk should be located in the /disks/ directory on the SD card, and should have a **.dsk** extension. Case doesen’t matter, so **.DSK** is also ok.

Create a temporary association

The 85StandardPac can be found here: “/EK_DISKS/DISKS0/85STANDARDPAC”

Copy the file using EBTKS commands

```
SDCOPY "/EK_DISKS/DISKS0/85STANDARDPAC","/DISKS/85STANDARDPAC.DSK"
```

Note: virtual floppy disk images are recognizable by their file length of 264 kB, exact size is 270336.

Note: the example SDCOPY also appended the required file extension.

(Cont'd)

Confirm the copy with

```
SDCAT /DISKS/
```

and expect to see a line with

```
85STANDARDPAC.DSK    ..... 270336
```

Copy the file using PC/MAC commands

Plug the SD Card plugged into your PC/MAC and use appropriate commands to copy 85STANDARDPAC from /EK_DISKS/DISKS0/ to /disks/ , and add the file extension .dsk

Now that the floppy image is in the right location

Once the file 85STANDARDPAC.DSK is in the /disks/ directory you can use the MOUNT command to associate it with an msus$ that has already been setup in CONFIG.TXT.

Note: you can’t create new msus$ while EBTKS is running.

```
MOUNT ":D302" , "/DISKS/85StandardPac.DSK",0

CAT ":D302"

[ Volume ]: StdPac
Name        Type  Bytes   Recs
MOVING      PROG    256     40
AMORT       PROG    256     17
POLY        PROG    256     29
SIMUL       PROG    256     47
ROOTS       PROG    256     19
CURVE       PROG    256     55
FPLOT       PROG    256     22
DPLOT       PROG    256     43
HISTO       PROG    256     36
TEACH       PROG    256     27
CALEND      PROG    256     22
BIORHY      PROG    256     21
TIMER       PROG    256     30
COMPZR      PROG    256     56
SKI         PROG    256     20
MUSIC       DATA    256     44
```

This is a temporary association. When the 85A (or any other Series 80 computer) is restarted, “:D302” will be associated with whatever is in CONFIG.TXT

The above MOUNT command could be re-issued, to re-mount the disk. You won’t need to repeat the prior copy operation.

Create a permanent association

This is done by editing the CONFIG.TXT file.

The copy step is optional, Create a temporary association

(Cont'd)

Appending the “.DSK” to the disk image file is optional.

Edit CONFIG.TXT and make the association permanent.
At about line 111, you should find a section that looks exactly like this:

```
1     {
2       "Comment": "msus$ 302",
3       "unit": 2,
4       "filepath": "/disks/Floppy_scr.dsk",
5       "writeProtect": false,
6       "enable": true
7     }
```

Change it to this (the only change is line 4)

```
1     {
2       "Comment": "msus$ 302",
3       "unit": 2,
4       "filepath": "/EK_disks/disks0/85StandardPac",
5       "writeProtect": false,
6       "enable": true
7     }
```

NOTE:

1. Only the “filepath” line (4) was changed
2. 85StandardPac did not need to be moved
3. Didn’t need the file extension .dsk to be added
4. Shows up in the boot screen messages if you scroll the text up
5. Association happens at boot time, no need for MOUNT
6. Permanent, until you edit this section of CONFIG.TXT

```
CAT ":D302"
```

will give the same results as in the prior section.

MOUNT: Additional reading

How to check the current file association to a msus$ MEDIA$

The mode flag for the MOUNT command MOUNT

To get a deeper understanding of this part of CONFIG.TXT, start reading at Disk Drives and read down till you get to the Printers heading.

If you want to be more adventurous with CONFIG.TXT, you really should start at the top of Working with CONFIG.TXT and read/understand everything up to the System Settings heading. Then read the secion relevant to task you want to do

Setting up EBTKS and the CP/M module

Apparently, EBTKS and the CP/M module can co-exist, and EBTKS can provide the disk services that CP/M requires. This was never a design goal.

This would not have been possible without the help of:

- Martin Hepperle: CPM-Usage.pdf
- All contributors to: Forum File Section for CP/M
- Donation of a CP/M module: TubeTimeUS on Twitter

```
61K CP/M Version 2.2
HP BIOS Version A   Revision 1.00

Copyright (c) 1982 by Hewlett Packard
Copyright (c) 1980 by Digital Research

A>
```

Using the Service ROM with EBTKS

This is discussed in the following forum article:

Using EBTKS to help with repair of Series 80 computers

Known Issues

- There’s something fisshy when you reference a non existent storage device which gives an error and then the Series80 computer and EBTKS both hang and need a power cycle
- Startup log messages on 85A/B screen is clipped for some items
- Current HP86/87 SD card image has :D301 assigned to an HP85A/B games disk
- No docs (or support) for WiFi
- Hp86/87 hangs if no SD Card plugged in

WiFi Support

WARNING!!! This document page is for software that is in development and is NOT ready to be used by end users.

- This page
- The firmware that runs on Teensy 4.1
- The software in the AUXROMs
- The firmware running on the ESP32 module
- The HTML served by the ESP32 module

are all in current development. Do NOT try following the procedures on this page if this warning is visible as you may permanently screw up your EBTKS WiFi support. Be patient, you will be notified by email when the WiFi services are ready to be used.

Requirements for WiFi connectivity

1. To utilize the EBTKS WiFi services, you will need a local network (LAN) that includes a WiFi router. This is usually either part of your home's internet connection hardware, as part of your cable/satellite/DSL/Dial-up Modem, or it is a stand alone Wifi router or access point connected to your internet modem via Ethernet. You will need the SSID (colloquially, the WiFi name) and the WiFi password
2. A Computer/Laptop/Smartphone that has WiFi connectivity for configuration
3. A Computer/Laptop that has LAN connectivity (via WiFi **OR** Ethernet) and runs at least one common internet browser for EBTKS access (Firefox, Safari, Chrome, Internet explorer, Edge)
4. A series 80 computer with an EBTKS installed

WiFi Overview

WiFi Modes

The Wifi functionality on EBTKS is implemented with an ESP32 WROOM32-D module which can be seen in the back right corner of the board next to LED1. It has a square metal cover. Internally in has a dual core Xtensa $^{®}$ 32-bir LX6 RISC CPU, a blizard of peripherals, 448 kB of ROM, 520 kB of RAM, WiFi and Bluetooth radios, and an integrated PCB antenna. Currently, EBTKS only makes use of the WiFi capability, and the CPU resources needed to support WiFi.

Mode 1

As shipped, the WiFi module is configured as a WiFi access point, broadcasting an SSID of "EBTKS_AP".

Mode 2

Giving the WiFi module access to your local WiFi router. You will need a Computer/Laptop/Smartphone that has WiFi connectivity, and the WiFi router's SSID and password

(Cont'd)

Mode 3

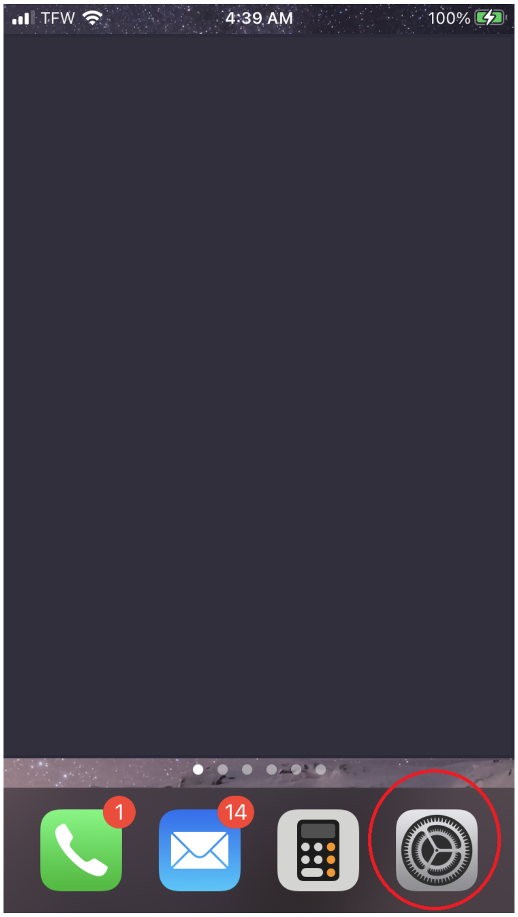
WiFi module is now seen as a URL by other local devices on your home WiFi network. The URL for EBTKS is **http://esp32_ebtks.local/**

Mode 4

Connected. You have opened a browser on your desk top or laptop computer, and entered the URL in Mode 3 into the address bar.

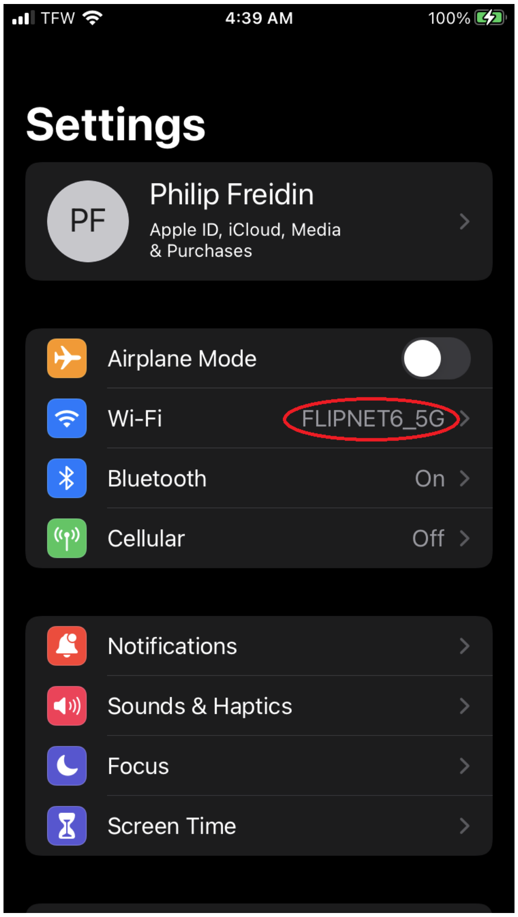
Getting from Mode 1 to Mode 4

The following example is using my iPhone, but the procedure will be very similar on any other device that can connect to WiFi and select the name (SSID) of the WiFi router. The text below refers to the areas marked with red text or ovals. Your Series 80 computer with an EBTKS installed should be turned on throughout this process.



Select the settings for your WiFi capable device.

(Cont'd)



TFW 4:39 AM 100%

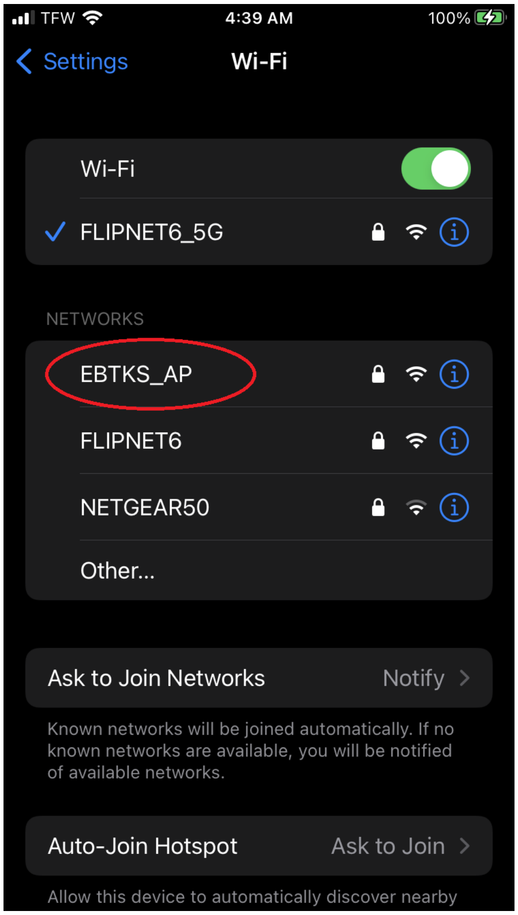
Settings

Philip Freidin
Apple ID, iCloud, Media & Purchases

Airplane Mode
Wi-Fi FLIPNET6_5G
Bluetooth On
Cellular Off

Notifications
Sounds & Haptics
Focus
Screen Time

Select the WiFi settings section. FLIPNET6_5G happens to be my normal WiFi router on my local LAN



TFW 4:39 AM 100%

Settings Wi-Fi

Wi-Fi
✓ FLIPNET6_5G

NETWORKS
EBTKS_AP
FLIPNET6
NETGEAR50
Other...

Ask to Join Networks Notify

Known networks will be joined automatically. If no known networks are available, you will be notified of available networks.

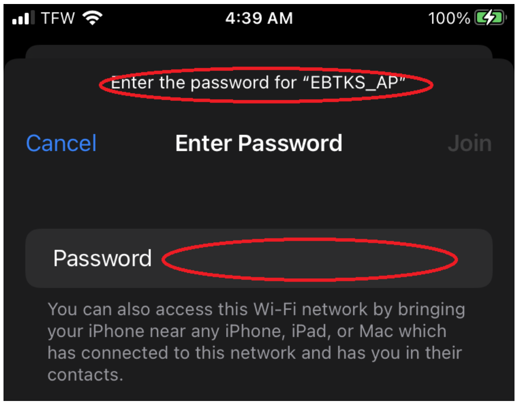
Auto-Join Hotspot Ask to Join

Allow this device to automatically discover nearby

You device should show your current SSID connection (in my case FLIPNET6_5G), and in the Networks section it shows other SSIDs that are also active in your location. The rest of this procedure assumes that you see the default SSID for EBTKS: **EBTKS_AP**

Select EBTKS_AP, so that you can connect to it rather than your normal WiFi router.

(Cont'd)

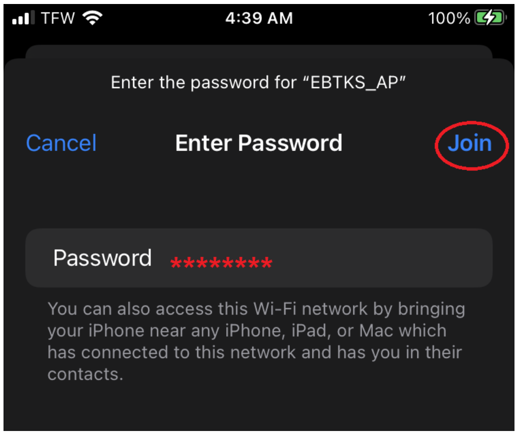


Enter the password for “EBTKS_AP”

You can also access this Wi-Fi network by bringing your iPhone near any iPhone, iPad, or Mac which has connected to this network and has you in their contacts.

You will be prompted for a super secret password to EBTKS_AP.

It is literally “password”

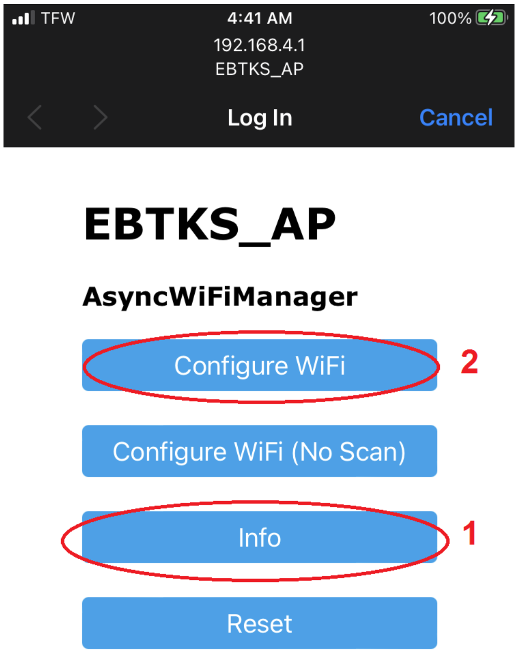


Enter the password for “EBTKS_AP”

You can also access this Wi-Fi network by bringing your iPhone near any iPhone, iPad, or Mac which has connected to this network and has you in their contacts.

After entering the super secret password to EBTKS_AP, join the “EBTKS network”.

Your device will now be connected to EBTKS, and you will be disconnected from the internet, since you have just disconnected from your normal WiFi router.

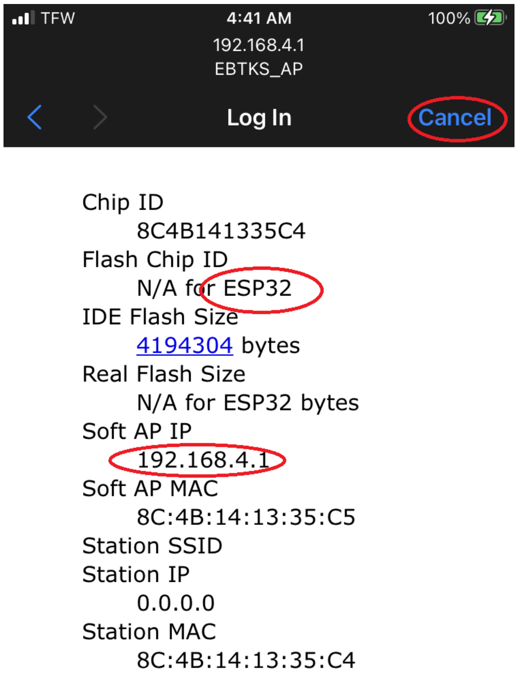


EBTKS_AP

AsyncWiFiManager

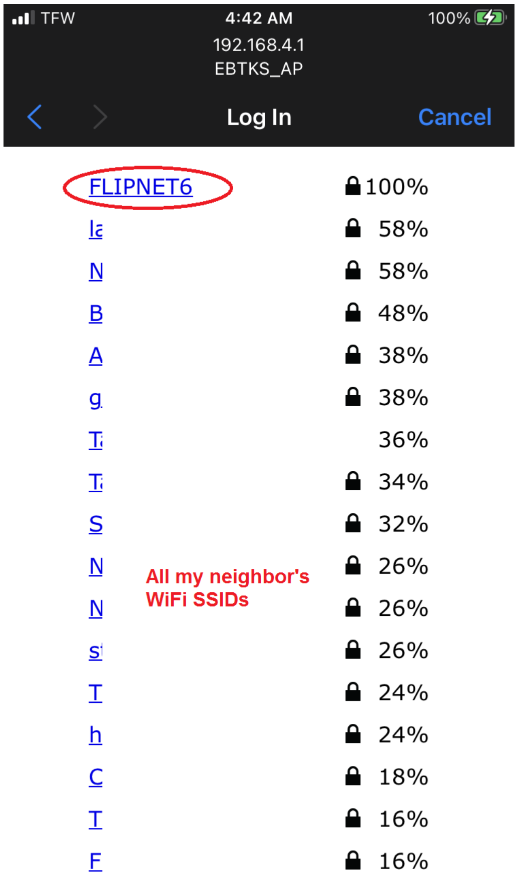
EBTKS will now present its WiFi management page. You may optionally select (1) “Info” , and you will see a view similar to the next image on this web page. If you skip (1), and instead select (2) “Configure WiFi”, just skip the next image and continue at the following one.

(Cont'd)



```
Chip ID
        8C4B141335C4
Flash Chip ID
        N/A for ESP32
IDE Flash Size
        4194304 bytes
Real Flash Size
        N/A for ESP32 bytes
Soft AP IP
        192.168.4.1
Soft AP MAC
        8C:4B:14:13:35:C5
Station SSID
Station IP
        0.0.0.0
Station MAC
        8C:4B:14:13:35:C4
```

This is the optional Info view that shows the status of your EBTKS WiFi access port. There is nothing you need to do on this page. When you have finished reviewing the information, select "Cancel" and you will be taken backto the previous view. Then select (2) "Configure WiFi"



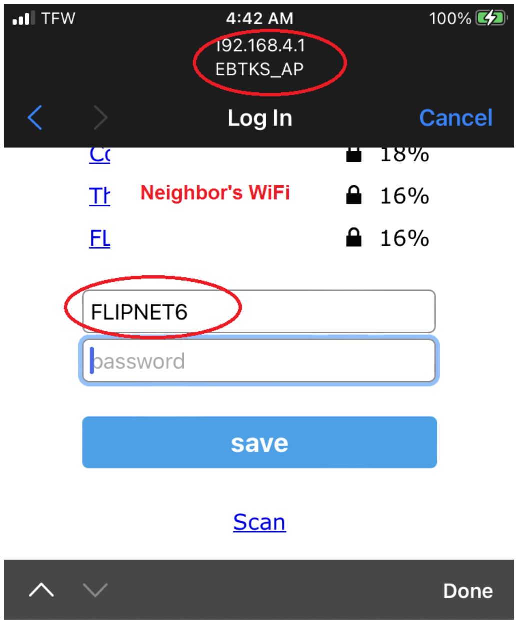
FLIPNET6	🔒100%
la	🔒 58%
N	🔒 58%
B	🔒 48%
A	🔒 38%
g	🔒 38%
Ta	36%
Ta	🔒 34%
S	🔒 32%
N	🔒 26%
N	🔒 26%
st	🔒 26%
T	🔒 24%
h	🔒 24%
C	🔒 18%
T	🔒 16%
F	🔒 16%

All my neighbor's WiFi SSIDs

This is the EBTKS WiFi Configuration page. You will see EBTKS_AP and "Log In" at the top of the page, and the rest of the page is all the SSIDs that are detected. Select the SSID for your Local WiFi router. (note, I have a dual mode WiFi routers, which presents as FLIPNET6_5G for 5 GHz operation, and also as FLIPNET6 for 2.4 GHz operation. EBTKS can only detect the 2.4 GHz SSID.) Select the SSID of your WiFi Router.

Note: As far as I know, any router that supports 5 GHz also supports 2.4 GHz, often with very similar names.

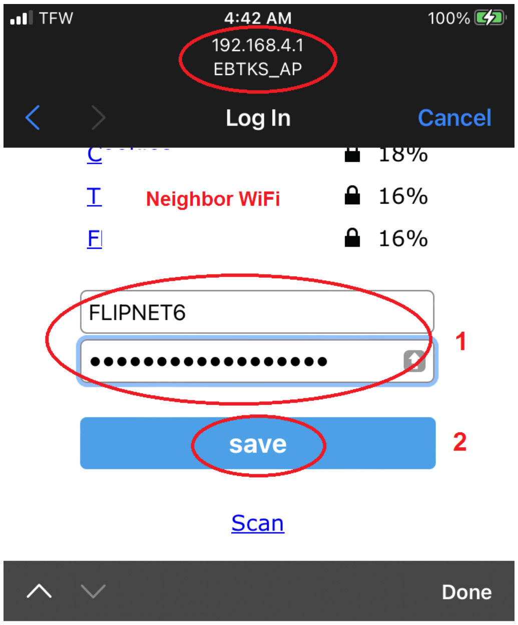
(Cont'd)



FLIPNET6

password

This step is where you tell EBTKS the SSID and password for your WiFi Router. You have already selected the SSID.



FLIPNET6

••••••••••••••••••

Now enter the password for your WiFi router. After entering it, select save. The information wil be saved in the EBTKS ESP32 WiFi module. This completes the configuration of the EBTKS WiFi Module.

You now need to go back to the Wifi configuration for your device. If you have been successful with the above process, you should navigate to the SSID selection screen for your device (third image on this page), and EBTKS_AP should NOT appear. Now select your normal Wifi router to reconnect to your local LAN.

Enable remote screen mirroring

For HP85A/B

To enable WiFi services on your EBTKS, the CONFIG.TXT must have the following settings, near the beginning of the file. Check the CONFIG.TXT file on your Micro SD Card, and make edits as needed to the "screenEmu" and "CRTRemote" fields. Both must be set to "true".

```
{
  "machineName": "HP85A",
  "CRTVerbose": true,
  "ram16k": true,
```

(Cont'd)

```
  "screenEmu": true,
  "CRTRemote": true,
  "tape": {
    "enable": true,
    "filepath": "/tapes/tape1.tap"
  },
```

For HP86A/B and HP87A/XM

(at the time of writing this documentation, screen mirroring does not yet work for these systems. The firmware update function should work.)

Set “screenEmu” to true and “CRTRemote” to false.

```
{
  "machineName": "HP87",
  "CRTVerbose": true,
  "ram16k": true,
  "screenEmu": true,
  "CRTRemote": false,
  "tape": {
    "enable": true,
    "filepath": "/tapes/tape1.tap"
  },
```

Connect your Browser to EBTKS

In your browser on a computer connected to your LAN (does not need to be connected via WiFi, Ethernet should also work) enter the following into the address bar:

http://esp32_ebtks.local/

Index

Note: If you open this PDF document with a web browser such as Google Chrome or Microsoft Edge, clicking any of the links below will take you to the EBTKS site.

A | **B** | **C** | **D** | **E** | **H** | **I** | **K** | **L** | **M** | **O** | **P** | **R** | **S** | **T** | **U** | **W**