Tiny-c "Load and Go" Version

for the

CP/M* Disk Operating

System

(January, 1979)

Note; This is not a tiny-c Manual. This is a supplement to the tiny-c OWNER's MANUAL, which is available separately.

*CP/M is a trademark of Digital Research, Inc.

Copyright, 1978, Tiny c Associates

Contents

Note: The disk has the interpreter source code in file TC.ASM. Also there are further improvements to the PPS beyond those described in C6. Two .DOC files on the disk describe these additions.

C1. Diskette Contents

The CP/M* diskette contains the following files:

```
TC.COM
PPS.TC
GETPUT.TC
ENDLINE.TC
CTC.ASM
```

It does NOT contain a copy of CP/M.

The version of tiny-c furnished is fully interfaced to the disk, and configured for an upper/lower case keyboard, and 24K of RAM. It can be reconfigured for an upper case only keyboard, and/or more RAM. It has been tested with CP/M version 1.4. It has not been tested with earlier versions.

C2. Normal Start with PPS

To start tiny-c and the PPS, first boot CP/M from another disk. Then mount the tiny-c disk in drive A and type:

```
tc pps
```

(Be sure there is one space between tc and pps, and type a carriage return at the end of pps.)

This will load and start tiny-c. Tiny-c then loads and starts the PPS. The copyright message and a single > will appear. Chapter 3 of the owners manual tells how to use the PPS.

C3. More on starting tiny-c

If you start tiny-c without giving a filename by typing just:

```
tc
```

it loads and starts tiny-c and enters the tiny-c system loader, (see the Tiny-c Owner's Manual, Section 6.3). This causes the copyright message and a triple >>> to appear. You can then load one or more tiny-c programs to run at system level, see Section 6.12.1.

Any file name given to the loader without a .filetype defaults to .TC as the filetype.

Alternatively, one file can be loaded by naming the file on the same line as the tc command. This is how PPS.TC is loaded in section C2 above.

*CP/M is a trademark of Digital Research, Inc.

C4. Leaving tiny-c

To leave a running application type the ESC key. To leave PPS, enter the command

```
.x
```

This returns you to the loader where the triple >>> is given. To leave the loader, and hence tiny-c, type another

```
.x
```

This will start a warm boot of CP/M from drive A.

The special character ^C for restarting CP/M is not implemented in tiny-c.

C5. Special Characters

The line kill character has been modified to NAK (cntl-U), and the DC2 (cntl-R) line echo has been added to conform to CP/M usage conventions. The other special characters are unchanged from the tiny-c standard. The complete list of special characters is:

ASCII name	Hex code	Function
DEL	7F	Delete one character
NAK	15	Cancel line typed so far
ESC	1B	Halt application program
DC2	12	Echo line typed so far

DEL is sometimes called RUBOUT. NAK is control-U on most keyboards. ESC (for escape) is usually a separate key labeled ESC. If not, find out what key on your keyboard generates 1B hex. It may be a control-key combination. DC2 is usually control-R.

C6. Modifications to PPS

There are two modifications to PPS. Control-R can be typed to echo a line as described above. This mod is actually in the library subroutine gs (get string) so will function for application programs using gs.

The directory of the currently logged, or specifically named disk can be listed using the command dir.

.dir	lists tiny-c files on the current disk, (.TC filetype).
.dir afn	lists files that match the ambiguous file name afn on the current disk.
.dir d:	Makes disk d current. d must be A, B, C, D, a, b, c or d.

.dir d:afn Makes d current and lists files that match afn.

afn is an ambiguous file name as described in the CP/M documentation, except that if a filetype is not given .TC is used as the filetype. *.* means all files. *. means files with blank filetype. * means files with .TC filetype. ? can be used in the same fashion as with the CP/M dir command.

C7. Disk I/O using the tiny-c library routines.

This version of tiny-c allows any number of files to be opened for processing. Unit 1 is built in. (See the Tiny-c Owner's Manual Section 2.9.2 for a discussion of input/output units.) All other units are addresses that point to an Extended File Control Block (EFCB), and you can declare as many of these as your program needs. An extended file control block is an array of 36 bytes. First we define the layout of this array. Then we show how they are used.

C7.1 Extended File Control Block (EFCB)

The EFCB is a CP/M file control block with three added bytes at the end. These are S, D, and NB in bytes 33-35 respectively.

S is the status: 0 for closed, 1 for reading, 2 for writing.

D is the drive: 0 for A, 1 for B, 2 for C, 3 for D.

NB is the next byte, used by the byte level access routines getbyte, and putbyte.

So the EFCB looks like:

```
0  1-8  9-11  12  13-14  15  16-31  32  33  34  35
et  fn   ft   ex    x    rc   dm    nr   s   d   nb
              Defined by CP/M <---- | --> Defined by
                                          tiny-c
```

An EFCB is declared as a character array, and given a meaningful name:

```
char master(35), update(35), newmaster(35)
```

Then the array names are used as units in the standard functions fopen, fread, fwrite and fclose:

```
int read, write, err
read=1
```

```
write=2
err=fopen(write, "a:master.dat", 0, master)
if(err)return
err=fopen(read, "b:update.dat", 0, update)
   .
   .
  etc.
```

C7.2 Filenames

A filename is a character string of the form:

```
d:fn.ft
```

with a null byte at the end. The d: may be omitted. The .ft may be omitted. Lower case alphabetics may be used, they will be case-folded to upper case to conform to CP/M file naming conventions.

If d: is specified the d must be one of the letters A, B, C, D, a, b, c or d. That drive is set into the EFCB. If d: is not specified, the currently logged drive is set into the EFCB.

The filename is blank padded and set into the EFCB. If an asterisk (*) is encountered in the filename, the filename is padded with question marks (?) and set into the EFCB.

If given, the filetype is set into the EFCB using the same padding and asterisk rules as the filename. If not given TC is used for the filename.

C7.3 Units and EFCB's

Notice that an EFCB need not be initialized as described in the CP/M documentation. This is done by fopen. All the programmer need do is provide the EFCB, and then use it as the unit in fopen, fread, fwrite and fclose, and take care not to change its contents.

The design of tiny-c requires there be a unit 1. It is used by the .r and .w commands of the loader and PPS. So a built in EFCB is furnished, and it is used whenever unit 1 is referenced. (This built in EFCB does not use the built in FCB provided at 0080H by CP/M.) Notice it must be opened using fopen just like any other file. But a 36 byte array need not be declared. Unit 1 may be used by tiny-c programs.

Any unit other than 1 is assummed to be the address of the EFCB used by the unit. So in effect units and EFCB's are the same thing. A file is "open" if there is an EFCB that is initialized that references the file. Note that any number of files can be open, but each requires a separate EFCB.

Once a file is closed using fclose, the EFCB is free and may be used for another file.

C7.4 Fread, fwrite and fclose

Once opened, the file can be processed using fread and fwrite. The maximum size block (record) is 128 bytes, not 256 as stated in the tiny-c Owner's Manual. Any record shorter than 128 bytes has a hex 1A placed in the first unused byte when written. When read, the returned length is the number of bytes before the first hex 1A in the record, or 128 if there is no hex 1A. This layout is compatible with many CP/M utilities for sequential files of ASCII text.

The layout can also be used for binary data. Full sized 128 byte records should always be written and read. When reading, ignore the returned length, as there will sometimes be a 1A hex in the binary data. This will cause the returned length to be garbage. But the full 128 bytes of data in the record are always read.

An end-of-file can only be caused by reading beyond the end of written data. A hex 1A in a record does not cause and end-of-file.

Fclose must be called after writing a file, and should be after reading. Fclose puts a zero in the status byte of the EFCB. If the file was being written, fclose calls the CP/M CLOSE to update the directory.

C7.5 Error Conditions

When an error is detected, either a filename string is printed, or the EFCB. Then one of these message is printed:

Improper drive -- If given, the drive designator must be A:, B:, C:, D:, a:, b:, c: or d:.

RW must be 1 or 2 -- In a call to fopen the first argument (RW) must be 1 or 2.

Cant find file -- The requested file does not exist on the designated drive.

Directory full -- Given by fopen when RW=2 and the directory is full. Unless especially arranged, a CP/M disk directory can hold only 64 file descriptions.

Read error -- Given by fread when FDOS could not perform the read.

Write error -- Given by fwrite when FDOS

could not perform the write. Usually the disk's data area is full, although this could also be caused by a crashed or bad disk.

Close error -- Given by fclose when FDOS could not close the file.

The filename string is printed as given in the name field of the fopen call. The EFCB is printed as:

xx d:ffffffff.ttt ex rc nr s nb

xx is the CP/M error code returned by the FDOS call, in hex. The other parts are drive, filename, filetype, extent, record count, next record, status and next byte.

C8. Calls to BDOS or FDOS

Three additional library functions are provided for setting up an FCB, printing it, and calling FDOS.

ifcb char name(0), fcb(32)
Initializes an FCB for direct calls on FDOS. Name is a filename string as described in C7.2. FCB is a 33 byte array as described in the CP/M documentation. ifcb can be used to initialize all but the last three bytes of an EFCB. Note that ifcb provides a default filetype of TC.

prefcb char efcb(35)
Prints an EFCB in the format described in C7.5. Does not print a message. Always shows a 00 in the xx position. Can be used to print an FCB. Just ignore the three bytes of garbage at the end.

fdos int fnum, arg
calls ENTRY location of CP/M (0005H) and performs the FDOS or BDOS function specified by fnum with argument arg. Returns the value returned by CP/M. This is the single byte returned in A except for function 27, for which the two bytes in A and B are returned as a 16 bit integer.

Private MC's 1001, 1002 and 1003 are used to implement these functions.

C9. Principal Memory Locations

The installation programs occupy addresses 100H to about 580H. The source code for the installation is included on the diskette in the file CTC.ASM.

Tiny-c starts at 600H. Use this as ORG in the address

calculations in Section 6 of the Owner's Manual.

Allocations as delivered, (all in hex) are:

name	as delivered	reallocated *
8080 stack	1800	
BSTACK	1800	
ESTACK	E580 =-1A80	
BFUN	1A80	
EFUN	E500 = -B00	
BVAR	1B00	
EVAR	DE10 = 21F0	
BPR	2200	
EPR	B310 = -4CF0	
Highest address used	4CF0	
Start of an application program	3685	

* Write in pencil in this column any reallocations you may do.

The dimension of PR is 5000 decimal. The highest used address (4CF0) allows the system to run in a 24K CP/M system with the Tarbell version of CP/M. Standard 24K CP/M owners can increase EPR to -51F0 (AE10), and the dimension of PR to 6280.

Those with more than 24K should re-allocate memory as described in the Owner's Manual, Section 6.

C10. Modifying the interface code

The source code for the CP/M interface between tiny-c and CP/M is furnished in the file CTC.ASM. It can be easily modified, for example to add a new private machine call. To generate a new TC.COM file, use DDT as follows:

```
A>ASM CTC          (assemble modified CTC)
A>DDT TC.COM       (load old tiny-c)
-ICTC.HEX          (implant new CTC into tiny-c)
-R
-^C              (leave DDT, warm boot)
A>SAVE 23 XTC.COM     (save 23 segments with new name)
A>XTC PPS          (test the new version)
```

Make sure the new CTC does not go past 0600H. If it does some of the code will have to be placed above 1800H, and the 8080 stack and other memory areas reallocated.

C11. Recovering from a crashed PPS session

PPS has been known to crash during an editing session. An unexpected DONE message is the symptom. The best defense is to use .w several times during a long editing session. But recovering from a crash is not too difficult.

After the DONE message, type .x to cause a warm boot. This will not destroy the program being worked on, unless it is very huge. Then load DDT alone, i.e. without loading any files. give the command

```
-mf,t,100
```

For f use the start of application address, e.g. 3685 for the delivered system. For t use the highest used address, e.g. 4CF0. Your tiny-c program is now located down at 100H. You're still in DDT so type:

```
-d100
```

to see that it is indeed there. Keep giving d commands until you find the end. Using the s (set) command, put a 1A hex into the first byte after the carriage return after the last good line to be saved:

```
text of last line   0D  1A
                          |
                  put 1A byte here
```

Then type control-C (C) to cause a warm boot. Use the SAVE command to write enough pages to a new file on disk. The easiest way to figure the parameter of the save is to convert from hex to decimal the page where you put the 1A byte. For example, suppose you set the 1A byte at address 1C7D. The first two digits are the page, 1C. Converting to decimal gives 16+12 = 28. So your save command is:

```
SAVE 28 XYZ.TC
```

Now you can restart tiny-c and PPS, read file XYZ and continue your work.

C12. Sample Programs

The file GETPUT.TC contains two byte level file access subroutines. These routines are not fast, but they are convenient for processing small files.

```
getbyte char efcb(163)
Returns the next byte from the file referenced by
efcb. Decimal 26 (hex 1A) is returned as the
end-of-file signal.
```

putbyte char c,efcb(163)
Puts the character c as the next byte of the file referenced by efcb. A decimal 26 must be the last character transmitted into the file.

Note the dimension of the efcb array. The first 36 bytes are the actual EFCB. The remaining 128 are a buffer to hold a segments worth of bytes. It is automatically written (or refilled) when full (or empty). The decimal 26 transmitted using putbyte causes the buffer to be written, even though not yet full. Also the written file must be closed using fclose, or the CP/M directory won't be updated.

The file ENDLINE.TC has a utility for inserting or removing linefeeds from a file.

endline char fromfile(0), tofile(0)
int linefeeds
Fromfile and tofile are filename strings as defined in C7.2. Linefeeds is zero or nonzero. The file fromfile is copied to the file tofile. If linefeeds is zero, all linefeeds are remove. If linefeeds is nonzero all linefeeds are removed, and exactly one linefeed inserted after each carriage return.

This program is useful for changing a file from the format produced by PPS, to that used by ED, or vice versa. For example

```
endline "pps", "ppslf", 1
```

produces a file called PPSLF.TC with linefeeds.

ENDLINE copies about 25 bytes per second using a 2Mhz 8080. Processing linefeeds into the PPS takes 5 minutes.