Max-10 Technical Reference

(c) 1988 by Dave Stampe

This reference is designed to supply programmers with information on the formats of Max-10 files, pictures and printer drivers. The data given in this document is as current as possible; but we can't guarantee that it won't change.

Max-10 files come in two types: ASCII (just text) which can be used to exchange text with other word processors and to edit BASIC programs; and **/CLP** (clipboard, or pasteable) files. This last type of file can be pasted anywhere in a document with the **Paste File** command. **/CLP** files are further divided into two types: Clipboards and Full Documents-- Full Documents contain headers, footers, and data about page numbers, etc. in addition to the actual document data.

ASCII Files can be loaded from just about any file that has ASCII characters imbedded in it. Characters are handled in the following way:

32 to 127- imported as ASCII characters
13- Return or End of Line character
9- Tab character
All Others- Replaced with character 127 {■}

A file saved with **Save ASCII** will contain only the ASCII characters **32** to **127**, tabs, and Return characters.

/CLP Files are much more complicated: they contain text with style and font tag pairs, pictures, rulers and page breaks. They also have a **data header** which contains data on memory size, number of paragraphs, etc. so that **Paste** can check if the operation will succeed before it begins. An optional data block on the end of the file contains the header and footer,as well as other data.

The Structure of a /CLP file

Header: contains several types of information. The first is 2 flags for word cut/paste support. These tell the **Paste** operation whether the start and end of the selection was on a word boundary, so it will know when to insert a space.

Next is data on the file's consumption of system resources, used for pre-**Paste** error checking. This data guarantees that a **Paste** operation will never exceed the amount of free memory, maximum paragraph count, or size of text paragraph.

The Header Format:

2 bytes-	Start and End word boundary flags. 0 if not on a boundary when **Cut** or **Copied**-- a word boundary is the same as the edge of highlighting when you double- click on a word in **Max-10**. If you are creating a full file that begins with a ruler, and ends with any full paragraph (not a string) you can set these to 0.
2 bytes-	Number of paragraphs in the document-- add 2 for safety. A paragraph is a text paragraph, a picture, ruler, or page break.
2 bytes-	Estimated memory (**NOT** file) size. To estimate this, add about 9 bytes per paragraph to the file size.
1 byte-	String Only flag. Set this byte only if the entire selection **Cut** or **Copied** came from one text paragraph, and did not include the Return at the end of the paragraph.
2 bytes-	String or first paragraph size (set it to 32 if first paragraph was not text)
2 bytes-	Last String size- set to 32 unless the last paragraph is a text string (the Return character was not included).

Data in a /CLP File:

After the header in a /CLP file comes the data. Each paragraph or string in the file begins with a tag to show what type it is: after that, the format depends on what the paragraph is.

/CLP Tags:

100-	End of Data (more data may follow in a full document file).
0-	Text (full) paragraph. A Return character is assumed to have been selected with the paragraph.
32-	String (partial) paragraph. No Return was Cut or Copied at the end of the paragraph.
1-	Picture paragraph. Data on the picture, its location and bit image follow.
2-	Page break paragraph- no more data is needed.
255-	Ruler paragraph- data follows.

All other tag values should be assumed to be reserved.

Exact Paragraph Formats:

End of data (100)- No more data needed. If any more follows, the data is assumed to be for a full document and is ignored when pasting the file.

Page break (2)- No further data is needed.

String (32) or Text (0)- The format for these is identical. When a Text paragraph is Pasted, a Return character is assumed to follow (even though there is none in the file). A 2 byte length count follows the tag; the byte count includes the start Font/Style tag pair.

Next is the starting Font/Style pair. This has the same format as other tag pairs, which are imbedded in the text which follows. The first byte of the pair is $A0+(the font number, 0 to 7). Then the style tag follows, which has the following bit pattern: 1,0,0,L,H,U,I,B; where B=bold, I=italics, U=underline, H=superscript, and L=subscript. H and L should never be set at the same time, and the tag codes above $B0 are reserved.

The paragraph text follows, with Font/Style tag pairs before each new text type. Tabs are usually character 9, although any value between 1 and 31 may be used. **Never use 0 in a text paragraph, as Max-10 uses this internally as its End of Text marker.**

Pictures (1)- Picture paragraphs include data on the position and size of the picture in the document, as well as data on its own size, and the picture's bit image follows. (Data on single- picture /CLP files is given later in this document).

2 bytes-	the size of the picture paragraph- use 20 + the size of the bit image in bytes.
2 bytes-	the left side of the picture in the document. This is given as a value from 8 to 576, and represents a horizontal position in the screen window (see Rulers for more data). Picture position and size are automatically clipped before printing.
2 bytes-	the vertical size of the picture in lines. The maximum is about 600- each line represents 1/72 inch on the printed page. Use 8 as a minimum.
2 bytes-	horizontal width of the picture on the screen. Each pixel represends about 1/80 inch on the printout. Use 570 as a maximum and 24 as a minimum.
2 bytes-	bit image vertical size in lines
2 bytes-	bit image horizontal size in pixels
1 byte-	length of a picture line in bytes. There should be at least 8 blank pixels or 1 byte added to the end of each line of the picture's image.

The picture's image now follows. There may be any number of bytes per line, as long as the last byte is blank. A bit in the image should be 0 if white, and 1 if black. Putting more than 550 pixels wide for an 80 dpi printer or 900 for a 120 dpi printer is a waste of space, as it will be compressed horizontally on the printout.

The maximum size of the bit image is 7660 bytes: if you exceed this, the end of the picture could get lost during editing. This is plenty of memory for a full PMODE 4 screen, but not enough for a 320x192 or 640x192 (HSCREEN 3) full screen. However, you could have a 560x100 letterhead (which would be printed at full resolution, but would take up nearly 8K in the text buffer!)

Rulers (255)- The ruler paragraph has a fixed length of 27 bytes, of which the last 2 are assigned the value of 0 and are reserved for future use. The rest of the data corresponds to margins, tabs, justification, and line spacing data. All numbers are in pixels from the left edge of the edit window, and are 1/80 inch per pixel. Use 8 as a minimum, and 576 as a maximum.

2 bytes-	Right margin
2 bytes-	Indent. This will be used as the left margin on the first line of each paragraph.
2 bytes-	Left margin. This is the left margin on all but the first line of a paragraph. It is also used to limit picture width, along with the right margin. If the Indent is to the left of the left margin, it also acts as a tab on the first line only.
8x2 bytes-	8 tab positions. They should be sorted from lowest to highest, and unused tabs should be at the end, and contain the value 40000.
1 byte-	0 if there are NO tabs to use, else 1
1 byte-	Justification type: 0=Full, 1=Left, 2=Centered, 255=Right.
1 byte-	Line spaceing: 0=Single, 100= 1 1/2, 200= Double. (You could use other values to fine-tune spacing of lines).
2 bytes-	Always 0 (RESERVED)

The Full Document Data

The Full Document File data follows the standard **/CLP** file body. It is ignored when a file is **Pasted** into a document, and is only used by the **Load...** command.

The data consists of several sections, each preceded by a special tag. Following is a list of the tags (in the order that they occur), and the data they precede. If a tag is not found in a file when **Max-10** expects it, the loading stops and default values are used instead.

101- Precedes the Header data.
102- Precedes the Footer data.

The Header and Footer data consist of 5 bytes of 0 followed by a series of tagged paragraphs identical to the main document, but without a Paste data header. It should end with the 100 tag.

Notes on Header and Footer: There should never be more than 5 paragraphs in a header or footer, including the first ruler. Don't exceed the limit of 6 screen lines, either.

103- Followed by general status data, designed so you can take up where you left off in editing:

4 bytes- Menu option data. If the corresponding menu entry should be checked off, use 128, else use 0. **Do not use any other values.** The 4 entries are: **Show Rulers, Show Header/Footer, Picture Perfect,** and **Key Click.**
2 bytes- The vertical position of the top of the screen in the document, given in lines (1/72 inch on paper). Note that the edit window is 180 lines high, so the maximum should be: **9** (inches) * **72** (lines per inch) * (number of pages in the document) **-180** (Edit window height). **In no case exceed 64000.**
2 bytes- First page number in the document. This should be between 0 and 999. (Line numbers above 999 are not fully printed in the header or footer).
6 bytes- For future use: these are (2 bytes each): the active page size (648), and the top and bottom margins (72), given in multiples of 1/72 inch. Note that these do not, at present, influence **Max-10**.

UPGRAGE VERSION ONLY:

104- Precedes the Extension file data. This is used for columning and other new options:

1 byte- Column number flag: 0= no columns, 1= 2 columns, 255= 3 columns.
2 bytes- Column offset in 1/80 inch.
1 byte- "Use char. 127 as space" flag: 0 or 128 **only.**

Notes on Files and Errors

Max-10 includes extensive error trapping in its disk routines. This is a discussion of what happens in various operations when an error occurs.

Save, Save ASCII: The usual error here is a condition where an old file would be erased, or disk space would be exceeded. **Max-10** checks before it saves a file, and brings these conditions to your attention. Files are saved with type of 1 and ASCII, so the ASCII output can be loaded with BASIC and most word processers.

Load ASCII, Load: If a disk error or a memory or paragraph table overflow occurs during the loading of a document, the paragraph containing the error will be deleted and loading will halt. This may also caused the loss of the header, footer and status data in a **Load** operation.

Paste: This is the most fragile operation. Even though the Clipboard is in RAM on 512K machines, the **Paste File** opeation still uses the disk. On most errors, **Paste** will simply abort and the paragraph may be lost. A text paragraph may be filled up with "d"'s at the end, or a picture may be filled with garbage. However, in a few cases, an error in **Paste** may corrupt the document, or even crash the computer!

Recovering Your Document

Max-10 was thoroughly tested before it was marketed; however, it is such a large, complex program that the possibility of problems cannot be completely ruled out. Also, in a few situations (a paragraph with many different font sizes falling across a page boundary, for example) screen updating may be somewhat incomplete. To reformat the screen, simply click above, then below the scroll box in the scroll bar to redraw the area you're working on.

During editing, font and style tags tend to accumulate in the text. The excess tags can be removed by saving and reloading the document-- **Load** automatically filters out unneeded tags.

If you do encounter an actual problem: this will be a rare case. In most cases, read the manual or do some experimenting to find why **Max-10** does what it does. However, these are some symptoms of a real problem:

-highlighting appears in the wrong line or outside the edit window
-garbage text appears at the end of a paragraph
-text cannot be deleted
-the computer "freezes"

In these cases, you <u>may</u> have a real problem. So: **Don't Panic.** If you have been saving backups regularly, you won't have lost much. Try saving the document and reloading it. Also, save an ASCII copy--even if you can't load the full copy, you'll have saved your text.

We need to hear about bugs so we can fix them. However, it's a waste of time just to say "There's a bug here somewhere". We need more information to go on. The first thing you should do is to try and duplicate the problem. If you can't tell us how to duplicate it, we can't fix it. Work at it till you find the minimum it takes to make the problem show up. If it only acts up on one document, send us a copy of that, too.

Creating Picture-Only /CLP Files

Max-10 imports pictures from the **Translator** and other programs through the **Paste File** operation and **/CLP** files that contain only one paragraph- a picture. We feel that this type of file is important enough to have a section to itself, as no doubt programmers will want to import pictures in sizes and formats that the **Translator** can't handle, or to convert **Max-10** files to other formats.

Pictures in **Max-10** are <u>always</u> sized when they are copied to the screen or printed: thus, the source image size is almost irrelevant to the displayed size. Of course, the picture looks best if compressed or magnified by a whole number (e.g. x1, x2 etc). Pictures appear about twice as tall compared to their width on the screen unless **Picture Perfect** is on; this is due to the resolution of the CoCo 3's screen. You can preset the destination size when converting a picture with the **Translator** program to ensure an even multiplication factor.

Also, since the screen has 80 dpi (dots per inch) resolution and some printers use 120 dpi, the picture may be stretched horizontally an additional 50% before printout to compensate. This means that x2 may look best on some printers.

A picture paragraph or file in **Max-10** consists of 3 data sections: first, the data for the displayed size in the document, then data on the source picture size, then the actual picture image. The total size (including picture and data) is limited to 7670 bytes, due to limitations in **Max-10's** memory managment routines.

To create a /CLP picture file, we need more than just the picture data- we need a "dummy" header to be compatible with the **Paste** operation. Only 2 bytes in this header actually depend on the picture data.

<u>The Dummy Header Format:</u>

2 bytes-	Dummy word start/end flags- use 0 for both
2 bytes-	Number of paragraphs- use 3
2 bytes-	Memory size- use the size of the picture's data plus 50
1 byte-	String flag- use 0
2 bytes-	First Text size: use 32
2 bytes-	Last Text size: use 32

Now the picture destination, source, and image data can be added. Be sure to terminate the file with a 100 tag!

1 byte-	Picture paragraph tag- use **(1)**
2 bytes-	the size of the picture paragraph- use 20 plus the size of the bit image in bytes.

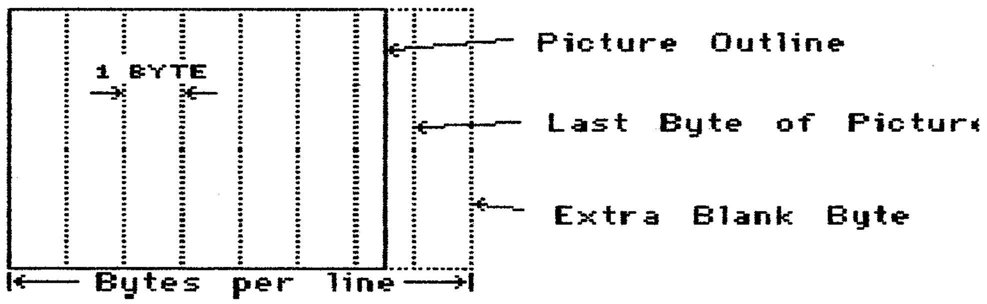
<u>Destination Size and Position Data:</u>

2 bytes-	the left side of the picture in the document. This is given as a value from 8 to 576, and represents a horizontal position in the edit window (see Rulers for more data). Picture position and size are automatically clipped before printing.
2 bytes-	the vertical size of the picture in lines. The maximum is about 600- each line represents 1/72 inch on the printed page. Use 8 as a minimum.
2 bytes-	horizontal width of the picture on the screen. Each pixel represends about 1/80 inch on the printout. Use 570 as a maximum and 24 as a minimum.

Source Picture Size Data:

2 bytes-	bit image vertical size in lines
2 bytes-	bit image horizontal size in pixels. The actual line will probably be longer: this value is used for computing sizing factors and to clip the end of the line.
1 byte-	length of a picture line in bytes. There should be at least 8 blank pixels or 1 byte added to the end of each line of the picture's image. The picture's image size is then (length of line in bytes)*(vertical size in lines).

The picture's image now follows. There may be any number of bytes per line, as long as the last byte is blank (0), and there is the same number of bytes in each line. A bit in the image should be 0 if white, and 1 if black. Putting more than 550 pixels wide for an 80 dpi printer or 900 for a 120 dpi printer is a waste of space, as it will be compressed horizontally when displayed or printed out.



The maximum size of the bit image is 7660 bytes: if you exceed this, the end of the picture could get lost during editing. This is plenty of memory for a full PMODE 4 screen, but not enough for a 320x192 or 640x192 (HSCREEN 3) full screen. However, you could have a 560x100 letterhead (which would be printed at full resolution,but would take up nearly 8K in the text buffer! You can also save a larger picture by making a file with several picture paragraphs, each of which is a vertical slice of the picture. Use the data above for each paragraph. This is the method used by the **Translator** for saving large pictures.

A Sample BASIC Picture File Creator

What follows is a simple BASIC program that will load a PMODE 4 picture with a black on white graphic in its upper left corner, find its size, and save it out to disk as a /CLP file.

```
100 POKE&HFFD8,0                          slow clock for disk access
110 PCLEAR4:ON BRK GOTO 200               break stops size scan early
120 INPUT "PIX FILE";N$
130 CMP:WIDTH 32:PMODE4,1:SCREEN1,1
140 LOADM N$                              load PMODE 4 file
150 POKE&HFFD9,0:AD=&HE00-1:LB=-1:LL=-1             fast clock
160 FOR L=0 TO 191:BF=0:FOR B=0 TO 31:AD=AD+1
170 A=PEEK(AD):POKEAD,NOT(A) AND 255:IF A<>255 THEN BF=1:IF B>LB THEN LB=B
        invert picture, count lines, find width
180 NEXT:IF BF THEN LL=L
190 NEXT
200 POKE&HFFD8,0                          slow clock for disk access
210 PRINT"LINES"LL+1
220 PRINT"BYTES"LB+1
230 ON BRK GOTO 0
240 BC=LB+2:LC=LL+1:PC=BC*LC              bytes/line,bit map size
250 PH=LB*8+8:PV=LC                       picture source h,v size
260 INPUT"FULL CLIP FILE NAME";N$         include /CLP in name!
270 OPEN"O",#1,N$                         open output file
280 X=0:GOSUB 500                         0,0 word flags
290 X=3:GOSUB 500                         dummy paragraph count
300 X=PC+32:GOSUB 500                     memory use estimate
310 X=0:GOSUB 550                         other dummy data...
320 X=32:GOSUB 500' EST. ADD TO TXT PARA
330 X=32:GOSUB 500' EST. ADD TO END OF TXT
340 X=1:GOSUB 550                         picture tag
350 X=PC+20:GOSUB 500                     picture size word
360 X=8:GOSUB 500'                        left margin
370 X=PV:GOSUB 500                        dest. v. size
380 X=PH:GOSUB 500                        dest. h. size
390 X=PV:GOSUB 500                        picture lines
400 X=PH:GOSUB 500                        picture width
```

(continued on next page)

```
410 X=BC:GOSUB 550  bytes per line in image
420 FOR L=0 TO LL:FOR B=0 TO LB+1:AD=&HE00+L*32+B
430 X=PEEK(AD):GOSUB 550          output bit map of picture
440 NEXT:NEXT
450 X=100:GOSUB 550 ' END OF CLIP FILE
460 CLOSE:END
500 'OUT WORD
510 X1=INT(X/256)
520 X2=X-X1*256
530 PRINT#1,CHR$(X1);CHR$(X2);
540 RETURN
550 'OUT BYTE
560 PRINT#1,CHR$(X);
570 RETURN
```

The following short program may be used to look at the structure of any file:

```
100 INPUT "FULL FILE NAME";N$
460 OPEN"R",#1,N$,1                 open file
470 FIELD #1,1 AS A$
475 FOR A=1 TO 30:GET#1,A           show first bytes of file
480 PRINTASC(A$);:NEXT
490 PRINTLOF(1):CLOSE:END           show file size
```

The Printer Dump Module

Max-10 uses a standard format for all its printer dumps. This is a machine language program, maximum size 512 bytes, beginning at $3300. It has its own RS-232 driver, and standardizes graphic and draft output on any printer.

Each module has a 4-byte header that has data on the printer, as well as double-strike and BREAK flags. The driver is entered just after the header, at $3304. Register B contains a value from 0 to 7, specifying which operation is to be performed. A and X may also be used as input or output by some routines.

Before I describe the driver, a few words about how **Max-10** uses it are in order. Draft (text) output just sends text and Return characters to the printer, and puts 1/2 inch top and bottom margins on fanfold paper. Graphic output is drawn in "slices" to a 56 line by 120 byte buffer in memory. This size allows for an integer number of print lines with 1, 7, and 8 "pin" printers. A byte in the header tells the print subroutine how many "pins" the printer uses. (By "pins" I mean the number of vertical dot rows printed with each pass of the print head.)

A graphic output leaves 1 inch margins at the top and bottom of standard 11 inch fanfold paper. For single sheets, no margins are used, so you can set them manually. This means that 9 inches of the page is actually used for output. The reason for the size of the margins is that some printers have trouble handling single sheets with smaller margin sizes.

However, the 56 line buffer does not evenly cover the page- it takes 11 full buffers plus 1 buffer with only 32 lines used. Thus, since a 7-pin printer will put out 35 lines on the last buffer, the bottom margin must be shortened by 3/72 inch. 8 and 1 pin printers are OK, but a separate bottom-margin eject code is used just in case.

"Setup" command codes cause the printer driver to send data to the printer to put it in a known state: scroll size, character width, etc.

A List of Driver (B Register) Codes and Operations:

0- **Text Mode Setup:** This should set up the printer to print text. It should set it for Pica (10 CPI) and a 1/6 inch scroll. **It should not actually feed any paper!**

1- **Graphics Mode Setup:** This performs any setup needed for Graphics output, such as scroll size, and, on some Tandy and Oki printers, the resolution (by setting the text size). **It should not feed any paper!**

2- **Text Line Feed:** This should feed 1/6 inch of paper (the standard 12/72 inch scroll). It must leave the printer in text mode.

3- **Graphics Line Feed:** This should feed a new line of paper for graphics: 7/72 for 7 pin, and 8/72 for 8 pin printers, for example. If you fed 1/144 or 1/216 inch of paper in double-strike between passes, you should use a different scroll for double strike, subtracting this amount, so that the page height is not changed. It must leave the printer in Graphics mode.

4- **Eject 1/2 inch margin:** Use this to eject 1/2 inch of paper for a margin: for example, you could use 3x1/6 inch line feeds, or a defined line feed of 108/216 inch. The printer should be left in text mode.

5- **Eject Bottom margin:** This also ejects 1/2 inch, but on a 7- pin printer should eject 3/72 inch less. This might be done with a defined- height scroll, or you could use 3x7/72 inch graphics scrolls and a 1/6 inch text scroll.

6- **Graphics Line Print:** This outputs one line of graphics (1, 7, or 8 "pins" high) with the top left of the source pointed to by register X. On exit, X should point to the start of the next line of graphics. For example, on a 7-pin printer, save the X register on entry, and add 7*120 to it before you exit.
If you are using a 120 dpi resolution printer, you should add a .35 inch margin (4 pica spaces do nicely). Double- strike should be supported (preferably with a 1/144 or 1/216 inch scroll between passes), and you should include some kind of "intelligence" so that the printer skips big white spaces at the right and left margins, and **especially** blank lines.

7- **Output Text Character:** This should simply output the value in the A register to the RS-232 routine. The X register should not be changed.

General Notes on the Printer Driver

The printer driver must observe certain conditions and precautions: First, the DP is $4000. This should not be changed, nor should this area be used- all scratch space must be inside the dump. The maximum size of the dump is 512 ($200) bytes, or from $3300 to $34FF hex. The interrupts must not be masked on exit. The PABORT flag should be reset on entry, and set only if the Break key is detected in the RS-232 routine. The clock speed is fast (1.78 MHz) and the Baud Rate variable at $95 is set to (the "normal" value)*2+5.

The following is an assembly listing of the printer- dump header and a sample RS-232 routine, including Break key detection. You may include these in your own drivers.

```
00100          ORG    $3300    STANDARD PRINTER HEADER
00110 RESOLV   FCB    1        0=80 DPI, 1=120 DPI
00120 LINEHT   FCB    8        # OF PINS ON PRINTER
00130 DSTRIK   FCB    0        1= DOUBLE STRIKE
00140 PABORT   FCB    0        SET ON EXIT IF ERROR/BREAK
00142 ENTRY    CLR    PABORT   RESET BREAK FLAG
00143          STS    ABORTS   SAVE STACK FOR BREAK EXIT
00144          JMP    INTERP   GO TO DRIVER INTERPERTER (B=opcode)
00145 ***** RS-232 OUTPUT ROUTINE ******
00150 PBYTE    ORCC   #$50     RS232 OUT
00160          PSHS   X,B      SAVE X
00170 BTEST    LDB    #$FB
00180          STB    $FF02    BREAK KEY TEST
00190          LDB    $FF00
00200          BITB   #$40
00210          BNE    GBYTE
00220          COM    PABORT   SET BREAK FLAG
00230          LDS    ABORTS   ABORT PRINTOUT
00240          ANDCC  #$AF     IF BREAK PRESSED
00250          ORCC   #1
00260          RTS
00270 GBYTE    LDB    $FF22    TEST BUSY BIT
00280          LSRB            IN PIA PORT
00290          BCS    BTEST
00300 TOUT     BSR    OUT1
00310          CLRB            OUTPUT START BIT
00320          BSR    OUTB
00330          LDB    #8
00340          PSHS   B
00350 BLOOP    CLRB            OUTPUT 8 DATA BITS
00360          LSRA
00370          ROLB
00380          ASLB            GET A BIT: 0 OR 2
00390          BSR    OUTB
00400          DEC    ,S
00410          BNE    BLOOP    LAST BIT?
00420          BSR    OUT1     OUTPUT STOP BIT
00430          ANDCC  #$AF     BE SURE INTERRUPTS ARE ON!
00440          PULS   A,B,X,PC RETURN TO CALLING ROUTINE
```

(continued on next page)

```
00450 OUT1     LDB     #2          1 OUT
00460 OUTB     STB     $FF20       (B) BIT OUT
00470          BSR     DELAY       2 X DELAY
00480 DELAY    LDX     >$95        DELAY LOOP
00490          EXG     A,A         DESIGNED FOR FAST CLOCK
00500 DLOOP    LEAX    -1,X        USE BAUD RATE= STD BAUD*2+5
00510          BNE     DLOOP
00520          RTS
00530 ABORTS   RMB     2           STACK SAVED HERE
```

The Order of Printer Driver Calls:

With some printers, the order of the printer driver calls may be needed to develop a printer dump. The sequence **Max-10** uses to perform draft and graphics printing follows:

Draft: (each page)

-2x#4 Top margin eject (unless single sheet is selected)

-For each line: #7 codes for each character, followed by 1 or 2 #2 text scroll codes.

-2x#4 Bottom margin eject (unless single sheet is selected)

Graphic Printout: (each page)

-2x#4 Top margin eject (#3 graphics scroll if single sheet is selected, to take the slack out of the printer mechanism)

-#1 Graphics setup code

-12 of: #6 Graphics line print and #3 Graphics scroll

-Bottom margin eject: #5 followed by #4 (unless single sheet is selected)

-#0 Text mode reset.

These are repeated for each page.

Structure of Fonts and Font Files:

The fonts used by **Max-10** differ from the fonts used by **Coco Max** in several ways. I will discuss the structure of the fonts and the font files, as well as precautions to observe if you wish to create your own fonts.

First, the fonts in **Max-10** are stored in memory, not on disk. This allows for very fast printing and screen updates, but limits the number of fonts. There is about 14580 bytes available for fonts in memory; this allows about 4 to 8 fonts, depending on the size of the fonts.

Also, **Max-10** does not allow resizing of fonts; so a different font is needed for each size. While this might be considered a drawback, the reason for this is the poor appearance of resized characters. All the fonts in **Max-10** have been hand-tuned for best printout quality: in fact, since many printers print out at 120 dpi (dots per inch) horizontal resolution compared to the 80 dpi resolution of the screen fonts, there are 2 versions of every font- an 80 dpi and a 120 dpi version.

These 120 dpi fonts are larger than the 80 dpi fonts, thus they limit the number of fonts that can be used at one time. The 80 dpi fonts are loaded on startup, and the 120 dpi fonts are loaded before printing, if needed. (With 512K of memory, both sets are preloaded at startup.) The fonts are loaded as a binary file block, and consist of individual fonts and a pointer table to each font. The 80 dpi file also contains the **Font** menu text.

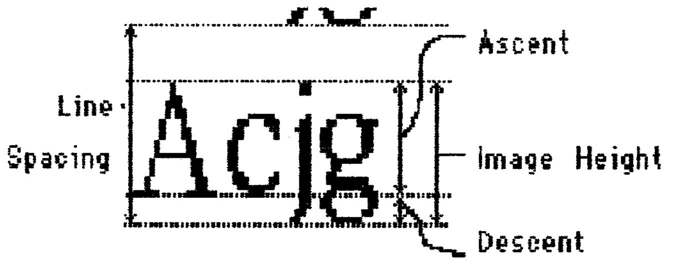
The font structure is designed to reduce size by 40% to 70% on large fonts compared to the MaxFont format. In a MaxFont, the bytes per line of each character in the font is the same; in a **Max-10** font, only as many bytes per line as are needed are used for each character.

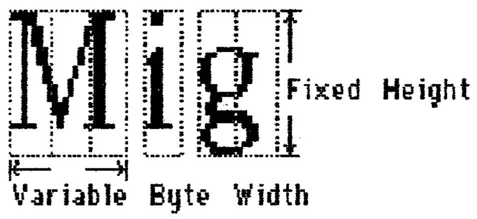
This requires a pointer table in the font, containing an offset in the image data to each character's image, and the number of bytes per line for each character. The font also contains a width table, giving the width in pixels of each character, and an 8-byte header, giving data for **Max-10** to use for line spacing, etc.

The detailed font structure follows:

The Header consists of 8 bytes:

1 byte-	The number of lines in the character image. This is the same for all characters in the font, and is the height from the top of the tallest character to the bottom of the lowest descender. **Max-10** does not shift a character to get a descender.
1 byte-	The number of pixels of "white space" to leave between each character.
1 byte-	The spacing between lines of text. This is the distance between the "baseline" (the bottom of capital letters) in each line. **It must never be less than 3+(the character image height).** A good rule of thumb is to make it about 1/3 greater than the character image height (given in the first byte of the header).
2 bytes-	Always 32 and 127.
1 byte-	The "lowercase descent" of the font. This value allows **Max-10** to compute the "baseline" of the font. A good way to find this is to count the blank lines below a capital letter in the font image.
1 byte-	Always 127.
1 byte-	The "highlight ascent". This can be figured as (line spacing) -(lowercase descent)-2.





The width table consists of 96 bytes- one for each character in the font. The numbers represent the width of the visible character; the gap between characters is specified in the header. Note that the width table and header in a 120 dpi font are a copy of those in the 80 dpi version of the font; during printing, the 80 dpi character positions are multiplied by 1.5 and the wider 120 dpi characters are "pasted" into the print line- this assures that there will be no differences between the screen display and the printed page.

The image pointer table consists of 2 bytes for each of the 96 characters in the font. The first 3 bytes are the number of bytes per line in the character's entry in the image table; the last 13 are an offset from the start of the image table where the first byte of the character's image may be found. For example, the address of a character's image can be figured as: (the pointer table entry AND $1FFF)+(3*96)+8+(the address of the font). This will take into account the sizes of the various tables, etc.

The character image table contains the actual image data for each character. A "1" bit is black and a "0" bit is transparent. There are the same number of "lines" in each character's image, but the number of bytes per line can vary, up to a maximum of 5. There must be at least 1 byte per line. The number of bytes used for each character is therefore (number of bytes per line)*(number of lines).

The Font File Types and Formats:

There are 2 types of font files, each consisting of a 120 dpi and an 80 dpi pair. The 80 dpi part also contains the menu text for the font. The 2 file types are individual fonts and font sets.

Individual fonts are files in pairs with the same name, but the filename extensions are "/80" and "/120" respectively. These are both binary files with 0 load origins, so they can be loaded at any address. The 80 dpi file also has, before the font itself, another 0 origin section containing the menu text entry for the font. This is the name of the font, folloewd by a zero byte, followed by a 1 or 2 character point size number, followed by another zero byte. The size of a font is assumed to be that of the largest part: the 120 dpi font.

The font sets are produced from individual font files by the **FONTASSM/BAS** program included on font disks. This program finds up to 20 font files on a disk, extracts the menu text and 120 dpi font size from each, and displays the data. You can then interactively select as many font as memory will permit, and then sort them in the order you want them to appear in the menu. The program will then assemble the 2 parts of the font set, and install them on your **Max-10** program disk or the font disk.

The name of the standard font sets that are loaded off the program disk at startup are **STDFONTS/P8** and **STDFONTS/P12** for the 80 dpi and 120 dpi sets, respectively. Note the file extensions and that the file names themselves are identical. You can specify either the default, standard file name or any other name when you create a new font set with the **FONTASSM** program.

Font set files are actual memory images of the fonts in **Max-10**. The fonts themselves load with an origin of $4000, and the first 32 bytes are a table of offsets to each of the fonts in the package. To get the font address, add the start address of the package to the table entry. Any unused entries should be copies of the first entry. Then, the fonts follow till the end of the package. The end should never exceed $78FF.

The 80 dpi package also includes a menu overlay that begins at $1105. The second byte is the number of entries in the menu, and the first is the bottom of the menu, computed from (number of entries)*12+9. The menu text for each font follows till the end of the menu. The end should never exceed $117F.

Note that if a document is displayed or printed with a diferent font set than the one it was created with, it won't look the same. In a document file, the fonts are specified in the font/style tags by a number from 0 to 15, which represents their order in the font menu and font table. This means that font order is important, too. So, be sure to keep a record of custom font sets with your documents!

Dave Stampe