Movie Credits

For The Atari 8 Bit Computer

800, 1200, 800XL, 130XE -- 40K Mem min

Version 2.0

LJK Enterprises, Inc.
1351 Yves Drive
Manchester, Missouri 63011-3669
(314) 527-6909

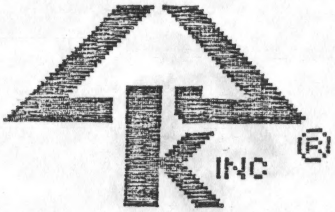
IMPORTANT

Make a backup copy of your Movie Credits Program disk using the Atari DOS copy program. Then place the original in a safe place and use the backup copy as your working copy of the program.

Because Movie Credits is supplied as a non-protected program, LJK will offer assistance, support and forthcoming updates only to registered owners of the program.

Failure to register the program on the enclosed registration form will forfeit all privileges of ownership including future updates and customer support.

MOVIE CREDITS



First Printing: May, 1991
Second Printing: October, 1991

LJK Part #2013

TABLE OF CONTENTS

LIST OF FIGURES

TABLE I

[CAPS]	Set lower case entry mode.
[SHIFT][CAPS]	Set upper case entry mode.
[ATARI]	Toggle input inverse flag off and on.
[BREAK]	Abort operation and return to outer level command.
[SHIFT][BACK S]	Cancel entire input line and start over. If in edit mode, the original line you were editing will be restored to you.
[ESC]	Exit entry of line by accepting only the characters that are to the left of the current cursor position.
[RETURN]	Exit entry of input line by accepting entire line.
[CTRL][A]	Erase from input line all from the cursor to the right.
[CTRL][B]	Go to the beginning of the line.
[CTRL][E]	Go to the end of the line.
[CTRL][F]	Find the next occurrence of the next typed character in the input buffer.
[CTRL][W]	Change the case of the character under the cursor; that is if it is upper case, it will make it lower case and vice versa.
[TAB]	Tab cursor over to next word. Tab will keep moving to the right until it finds a space character.
[SHIFT][>]	Toggle Insert mode on and off.
[CTRL][>]	Insert one character (space) at cursor location.
[CTRL][BACK S]	Delete character at cursor location.
[BACK S]	Delete previous character typed into input buffer.
[<-]	Move the cursor back 1 space (non-destructive backspace).
[->]	Advance the cursor by copying over the character from the screen and adding it to the input buffer.
[↑] (Up arrow)	Move the cursor up. This will not change your position in the input buffer, but will only move the cursor up.
[↓] (Down arrow)	Move the cursor down. Again, this will not change your position in the input buffer, but will only move the cursor down.
[CTRL][2]	Clear entire screen
[CTRL][TAB]	Clear screen from cursor location to end of screen.
[SHIFT][TAB]	Clear screen from cursor location to end of line.
[CTRL][I]	Move Cursor to top left screen position (home).

Special Keys

[CTRL][:]	Generates a ` (\$60).
[CTRL][;]	Generates a copyright symbol (\$7b).
[CTRL][<]	Generates the corner arrow (\$7d).

Introduction

Congratulations on your purchase of LJK Movie Credits. Movie Credits will allow you enhance your videotapes by adding header and trailer titles made up of a combination of text and graphics.

Graphic Generation

LJK Movie Credits generates graphic images on a 160 by 96 pixel four color screen. The screen uses a cartesian coordinate system with the horizontal locations being the x axis and the vertical locations the y axis. The upper left of the screen is location x=0 and y=0 (0,0) and lower right of the screen is x=159 and y=95 (159,95). This screen is demonstrated in Figure 1:

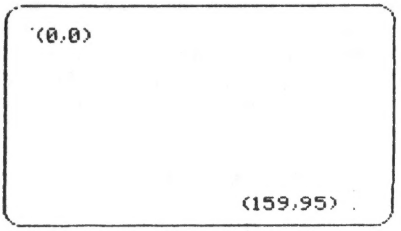


Figure 1

There are two different screens to allow you to prepare a screen on one video page while viewing the other one for smooth transitions. Graphics are generated in four different ways; they are: 1.) proportional character fonts; 2.) predefined 80 by 48 bit images; 3.) combinations of patterns and blocks of 8 by 8 character grids layed out on a 20 by 12 work area; and 4.) actual graphic primitives such as arcs and lines. Each of these graphic modes has modifiers to enhance their output. All can be run in double horizontal, double vertical or double both modes (size). Characters can be printed in solid, shadow or outline, they can also be modified by italics and/or boldface.

Creating Programs

A Movie Credits program is made up of a series of commands and parameters. Commands can be on separate lines or on the same line (generally, we separate them with a space, but no delimiter is required). Some commands **must** be the last command on the line such as the string prints and the comment.

We create our programs in the editor which is a line oriented standard text editor. We reference lines by line numbers; but the line numbers are only reference and are not a part of the file. In fact, the line numbers for a particular line will often change with insertions and deletions.

Our objective in creating a Movie Credits program is to move our cursor position around to where we want it, generate a graphic of one of the four types mentioned above, pause for the viewer to read it and to repeat the process again and again. An alternate method of entry will also be described for the static portions of your screen information. This will be done with a "What you see is what you get" type screen builder.

The Manual

This Manual is divided into three parts, an edit session, a tutorial and a reference section. The edit session and the tutorial are meant to be gone through while **sitting at the computer and doing the exercises**. The reference section goes into more technical detail of each command. Please go through the edit session and the tutorial sequentially. They only take a little time to execute and will greatly enhance the ease of slipping into Movie Credits.

You should also find the sample files on the program disk to be of use to you. You can edit the text on them to suit your needs and have a ready made library of different file types.

The Hookup

The hookup can be made by simply selecting the output of your computer switchbox to go to your vcr rather than your television set. A more sophisticated approach is to take the video output from the computer into the video input of the vcr. Some vcr's have a switch for using input from the

tuner or the camera/external/remote. If your machine has such a switch, you need to select the external for the program and the tuner when on tv. Most vcr's will select the external if a cable is in the video jack. On vcr's without a switch, the cables must be plugged and unplugged to go between the computer and the vcr tuner. **Important:** when in the computer mode using the video input jack to the vcr, you must have something in the audio jack as well. This could be an external audio source for music or narration and the like. Things can get even more sophisticated with the addition of video editors and/or mixers (if you are that stage, you should, by now, have an idea of how your setup should go). A typical setup might look like Figure 2:

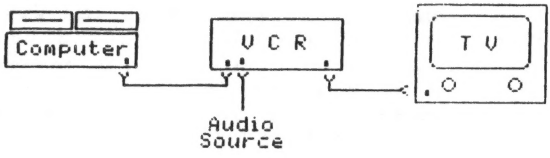


Figure 2
The Hookup

Starting The Program

The program is started by placing the diskette in drive 1 and turning on the machine. The program will boot up and load three files. The files are the block characters (MC.BLK), the graphic bit patterns (MC.GR) and the proportional fonts (MC.FNT). If for some chance there is an error loading any one of these files (not on the disk or an i/o error), you can recover from it by bloading the file later on.

Movie credits is free form in input (spaces are ignored and multiple commands per line are allowed). Commands can be in either upper or lower case or a combination thereof. Filenames will all be converted to upper case. Since strings often contain mostly lower case, it generally is advisable to run in lower case mode. This can be achieved by pressing the **[CAPS]** key by itself.

Part I - Line Basics

When we first boot up Movie credits we are greeted with a prompt of 'MC:' and the cursor is there waiting for us to type something. We are in the Movie Credits editor. From here we type our commands to manipulate our text. Let's start by typing **help [return]**. We see two sets of commands on the screen. The second set is the movie credits commands and we will not worry about them at this time. The first set, however, is the editor commands that is broken up into two sections, the input/output commands used for loading and saving information and the edit commands for manipulating the information.

Any time we type a line of input, we have a series of editing keys that help us generate the input line we want. These keys are described in detail in Table 1 in the reference section. Let's type **Now is the time for** but don't press [return] yet. We can now go to the beginning of the line by holding down the [CTRL] key and pressing the [B] key (do this now). You see our cursor has moved to the N in Now. We can insert a bunch of characters by holding down [SHIFT] and hitting the [INSERT] key. We now type **Of course it is** ; we no longer wish to insert and can go back to change mode by hitting [shift][insert] again. We no longer want the N capitalized on Now. We can change the case of it by hitting [CTRL][W]. We can advance a word at a time with [TAB] and to the end of the line with [CTRL][E]. These keys and all the others listed in Table 1 are at your beck and call whenever you are entering a line of input, including filename entry under the file section of the Edit RUN command. We should now press [SHIFT][DEL] to cancel our sample line.

Part II - Basic Entry

To ensure we have a clean editor, we type the command **new** and press return. We are prompted to **Verify?**. Typing a [Y] will continue the operation; an [N] will abort it. This verification is required here because new cannot be revoked once implemented. The file in memory will be gone. We answer this time with a [Y] and we come back to our MC: prompt.

We now wish to add information. We do this with the ADD command. This command is used so often, we have a single

letter abbreviation of A. So now we press **A [return]** (We could also have typed add [return]). We now see to the left of the cursor the number 1. This is our reference line number. As we are starting a new file, we begin with line 1. Now type in **[Edit Example]** and press return. We now are prompted by 2. We have stored line 1 and are ready for input of line 2. To exit the add mode, you simply enter no line (by pressing [return]).

We can now examine our file. This is done with the list command. We type **list** or the abbreviated version **l**. Our file (all of one line) is displayed on the video screen. If our file had been of considerable length, we may have wished to display only a portion of the file. We could do this by the reference line numbers. We could say:

```
list a          Lists only line a.
list a,b        Lists lines a through lines b.
list ,b         Lists 1 through line b.
list a,         Lists a through the end of file.
list a-         Lists 20 lines starting at a.
```

To add more text to our file, we simple do the add again. This time the line number prompt will be a 2 rather than the one the first time we entered add.

We now get our file to this point:

```
 1 [Edit Example]
 2 font(4)
 3 c"FBI Warning:
 4 font(0)
 5 cr c"Federal Law provides severe
 6 cr c"criminal penalties for the
 7 cr c"unauthorized reproduction or
 8 cr c"distribution of Copyrighted
 9 cr c"motion pictures and video
10 cr c"tapes (Title 17, #501 & #506).
11 cr c"This felony is punishable by
12 cr c"up to 5 years in prison
13 cr c"and/or a $250,000 fine.
14 wait(0) wait(0) reset
15 q
```

When we have reached this point, we test our little program (yes, that's what we are creating!) by typing **TEST [return]**.

We notice that the line starting with distribution is after a blank line. This is caused by line 7 of our example taking up exactly the entire line and creating its own cr (carriage return). To alleviate this problem, we simply remove the added cr in line 8. We do this by typing **edit 8** or **E 8**. We now see line 8 listed on the screen with the cursor past the 8 on the c in cr ready to edit the line. For our case, we can simply hit [CTRL][DEL] for 3 times to remove the cr and the space after it. We then hit [return] which accepts all the rest of the line. When we test our example now, we see that the offending blank line has been removed.

We now wish to further enhance our little program by setting off the FBI Warning: even more. We are going to change the color of it and change the style to a bold faced type. We must also remember to remove the bold face and go back to our regular color after we print that line. We now edit lines 2 and 4 to read:

```
2 font(4) color(2) style(8)
3 c"FBI Warning:
4 cr font(0) color(3) style(0)
```

We accomplish this by typing **e2,4**. When we get to line 2, we type [CTRL][E] to get to the end of the line and add our color(2) style(8) then hit return. When line 3 comes up, we simply hit return to keep it in its current state. On line 4 we first insert a cr space in front of the font(0) by hitting [SHIFT][INS] typing the cr space and then hitting [SHIFT][INS] again to remove the insert mode; we again use [CTRL][E] and add our text. When we test this (**TEST [return]** again) we see the FBI in a bold face green color.

To change our color scheme to something more dramatic we will insert a new line before any of our program commands. We now type **Ins 2** or **I2** and type the following line:

```
2 pen($28,$4e,$94,0)
```

When prompted for the next insert line (with the 3), we simply hit [return] to exit insert (this is just like we exited add). If we list our file now, we will see that all the lines have been moved up one after line 2 to make room for our new line. The file now looks like:

```
 1 [Edit Example]
 2 pen($28,$4e,$94,0)
 3 font(4) color(2) style(8)
 4 c"FBI Warning:
 5 cr font(0) color(3) style(0)
 6 cr c"Federal Law provides severe
 7 cr c"criminal penalties for the
 8 cr c"unauthorized reproduction or
 9 c"distribution of Copyrighted
10 cr c"motion pictures and video
11 cr c"tapes (Title 17, #501 & #506).
12 cr c"This felony is punishable by
13 cr c"up to 5 years in prison
14 cr c"and/or a $250,000 fine.
15 wait(0) wait(0) reset
16 q
```

Part III - Move and Copy

Let's now take our sample file and add the following to it:

```
17 swap page(1)
18 pos(0,0) pattern(20,3,0)
19 page(0) repeat(10)
20 display(1) wait(50)
21 display(0) wait(50)
22 again
```

When we test this example, we see no change in our screen! What has happened is we have added all of this after the q in the file telling us to quit execution. We need to move this information to above the q (in fact it should be above the waits and reset). We now move this information by typing **Move 17,22;15** or **M 17,22;15**. This will take our new lines (17 to 22) and move them in the file to insert them **before** line 15. Our newly moved file will look like this:

```
 1 [Edit Example]
 2 pen($28,$4e,$94,0)
 3 font(4) color(2) style(8)
 4 c"FBI Warning:
 5 cr font(0) color(3) style(0)
 6 cr c"Federal Law provides severe
 7 cr c"criminal penalties for the
 8 cr c"unauthorized reproduction or
```

```
 9 c"distribution of Copyrighted
10 cr c"motion pictures and video
11 cr c"tapes (Title 17, #501 & #506).
12 cr c"This felony is punishable by
13 cr c"up to 5 years in prison
14 cr c"and/or a $250,000 fine.
15 swap page(1)
16 pos(0,0) pattern(20,3,0)
17 page(0) repeat(10)
18 display(1) wait(50)
19 display(0) wait(50)
20 again
21 wait(0) wait(0) reset
22 q
```

Had we wished to make a copy of those lines instead of just moving them, we would have used the copy command which has the same form as move (i.e. **COPY 17,22;15**). As with the move command, a copy of lines 17 through 22 would have been placed before line 15; but in this case, the lines would also have been left in their original locations as well. Since the copy command is used less frequently than the move command, we have not allowed the shortcut single letter C as a substitute for the full copy.

Now when we test our example we have the FBI portion of the screen flashing for 10 times before settling in. We have now placed our code in the proper section of the file by moving it there.

Part IV - Filing Away

We now wish to save our program that we worked so diligently to prepare so that it won't be lost. We could save it on the program disk, but that is not a good idea as it would quickly fill up and we'd have no place to go. Instead we wish to format a new disk to prepare it to accept information.

We first perform the **STAT** command. This will tell us which drives are available, their current density and the size of our file as well as the free space remaining in memory. Do this now. Your screen should look something like this:

```
Drives: 1S2D3D
 Length:   641 $0281
  Free: 12670 $317E
```

You will notice the file length and the free space are given first in decimal and then in hex (preceded by the $). The drive shown in inverse is the current drive. We can change drives with the **DEV** command followed by the drive number and, optionally, either and S or a D to signify setting single or double density (if no letter follows, the current density will be used). If you have a system that support double density, you should probably be in that mode and should execute the command **dev 1d.** We can now place a blank disk in drive 1. Be sure it is blank because all information on it will be lost. We can now format the disk. In addition, we will give it a volume name of up to 15 characters for identification. After placing the blank disk in drive 1, we now type **FMT File Disk #1** and press return.

Assuming no errors have occurred we can now view our disk by typing **dir** [return]. You will see the volume name File Disk #1 and 707 free sectors. The dir command can also be used with an optional file name identifier where ? represents a single character wildcard and * represents a block of 0 to n characters. Using dir without parameters is the same as dir *.*.

We are now ready to save our file. We type **Save fbi.lib** [return]. The file from memory is now copied to the disk for later retrieval. You will notice we used the extension (second half of the filename after the period) of lib for this file. That is because this little bit may well be used on several different tapes as an introductory piece. When we retrieve things we may be putting together several pieces in a cut and paste type affair to create our new file for the particular video tape we are working on. When we come back to retrieve we will start with a load that takes the form of **Load filename** which clears memory and loads a file at the beginning of our edit buffer and can add new files to the end with the apnd command (apnd filename).

We now check our file by doing a dir and there it is on your file disk. Let's take a minute here to discuss three other file commands. The erase command takes the form of **ERASE filename** which is used for deleting files from disk. Wildcards can be used here as well. The Bload and Brun command are used to access binary files. As mentioned in the introduction, if for some reason there was an error loading one of the three files (block, graphic or font) at boot time, this command will get that file in memory (they are kept as files for flexibility in changing them later on

without changing the program). When a file is bloaded by the command **BLOAD filename** you are shown the address at which the file loaded and the length of the file. For example, if from the program disk we typed **bload mc.blk** [return] our screen display would look like:

```
Address: 11264 $2C00
 Length:  1024 $0400
```

To load and execute a binary file, we use the **BRUN filename** command. These must not only be binary files, but must be executable code files rather than data files as the font files are. Typical examples would be the graphics editor, the character editor and the screen builder (these programs may or may not yet be available as adjuncts to your Movie Credits system). These binary files usually have the extension .COM to them to indicate command files.

Part V -- Search and Replace

A further enhancement to the editing of files are search and replace. We have three commands for this: **FIND lno,lno "sstr"** which will show on the screen all occurrences of the string sstr in the range of line numbers specified; **REP lno,lno "sstr"rstr"** (rep could be abbreviated r) which will globally replace the string sstr with rstr in the range of line numbers specified and **SRCH lno,lno sstr,rstr** which will act as the replace with the exception you will be prompted to verify each change with a Y or N.

Strings are optionally delimited. That is they normally are delimited by commas(,). If you wish a comma in your string, you need to delimit it by either a" or a '. This is also the case if the string starts with a space or a number (the program will thing it is a line number). Examples of string delimiting are:

```
LOAD file.src          String is file.src
FIND   mylife          String is mylife
F long, long str       String is long (, delim)
F "long, long          String is long, long
R 12,35 " me" mine     search string is " me",
                       rep string " mine"
```

Going back to our file, we wish to have the fbi warning given only at the start of our tape. To make this a truly universal library tape, each line must begin with the word head (so that it will only execute when we are doing the tape header). We could just edit each line and insert the word head with a space after it for readability in each line. But, you will notice a great deal of lines start with a cr. To ensure we do not get in trouble with the word criminal (which starts with a cr) we type **rep cr ,head cr** making sure to include a space after the second cr. This changes several of our lines, but not all of them. We notice a number of lines start with c", we can then change them but should use the srch command as many other places the c" is used. We now type **srch c",head c"**. When prompted for the change, we are shown the changed line and asked to verify. The first time is line 3 and we answer yes, the second time is line 6 which already has a head and now will have two heads so we answer that one no. We continue on like this through the file, verifying the changes to make and canceling the others.

We still have some lines to change to insert the head at the start of the line. Some we can do with rep (lines 18,19 with display), others are just as easily done with editing the lines.

When we have completed all our changes, the file should now look like:

```
 1 [Edit Example]
 2 head pen($28,$4e,$94,0)
 3 head font(4) color(2) style(8)
 4 head c"FBI Warning:
 5 head cr font(0) color(3) style(0)
 6 head cr c"Federal Law provides severe
 7 head cr c"criminal penalties for the
 8 head cr c"unauthorized reproduction or
 9 head c"distribution of Copyrighted
10 head cr c"motion pictures and video
11 head cr c"tapes (Title 17, #501 & #506).
12 head cr c"This felony is punishable by
13 head cr c"up to 5 years in prison
14 head cr c"and/or a $250,000 fine.
15 head swap page(1)
```

```
16 head pos(0,0) pattern(20,3,0)
17 head page(0) repeat(10)
18 head display(1) wait(50)
19 head display(0) wait(50)
20 head again
21 head wait(0) wait(0) reset
22 q
```

A final finishing touch would be to remove the q so that other things could be appended to the file (true library file). This can be done with the DEL command. We type **del 22** [return].

We now resave our file by typing **save fbi.lib** [return]. This will replace the old version of the file. To test it, if we just type test (since we have always been in trail mode, we are still in trail mode) we must use the run command which prompts us. You will see using trail does nothing (all commands are head commands) and head gives us our warning message.

When we type run, we are prompted with Head, Trail, File or Quit? Pressing H will yield our warning message, pressing T in this case will do nothing as we have no trailer code and pressing F will cause a directory of the current disk drive to be displayed prompting for a file to load (this option is used to change files when creating multiple Movie Credits titles) and Q will return us to our edit mode and the MC: prompt.

Part I - Text Graphics

A Movie Credits program is made up of a series of commands that can be on one line or on several lines (you may have multiple commands on a line). Some commands such as the comment ([) and the string print require they be the last command on the line.

The Movie Credits screen is a cartesian coordinate system of 160 horizontal locations and 96 vertical locations. The horizontal locations are known as the x axis and the vertical locations are known as y coordinates. The upper left of the screen is location x=0 and y=0 (0,0) and lower right of the screen is x=159 and y=95 (159,95). We can position the cursor by absolute commands of xpos (for the x axis), ypos (for the y axis) and pos which does both x and y. We also have a relative position of rpos for both the x and y axis.

When we print strings, we can print them where the cursor is (with "), centered on the x axis (by c") or aligned with the right margin (by r").

From the editor generate the following text:

```
1 cls "Our
2 ypos(30) c"Summer
3 ypos(60) r"Vacation
4 wait(0) q
```

Now type **test [return]**. The screen should appear as it does in figure 3. If it does, congratulations, you have just completed your first movie credits title!

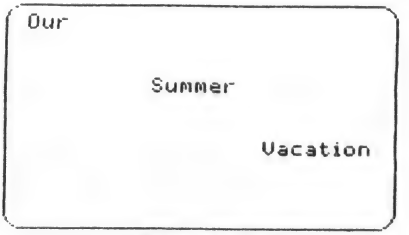


Figure 3

Type list from the editor. Line 1 starts by clearing the screen (CLS) which also does an absolute cursor position of x=0, y=0. Then a string is printed at the current location (0,0) with the (") command. The string generated is "Our" (all strings under Movie Credits go to the end of that input line). In line 2 we moved the cursor to a vertical position of 30 (ypos(30)) and generated a centered string of Summer with the c" command. Likewise, line 3 generates the string Vacation to the right of the screen at a vertical location of 60. Line 4 tells us to wait for a time of 256 (0=256) increments before continuing and the q tells us to stop.

Now lets enhance our program by having a delay between the printed of each of the strings to the screen. We now edit lines 2 and 3 to read:

```
2 wait (50) ypos(30) c"Summer
3 wait (50) ypos(60) r"Vacation
```

Now when we test our program (by typing test [return]) we find a delay between each string printed.

To enhance our program a little further, let us remove the irritating sight of the string printing. We can do this by building the string on graphics page 1 while looking at page 0 and then swapping it into our viewing screen. We do this by inserting before line 1 the line: Page(1) and then changing lines 2 and 3 to start with a swap (before the wait(50)). Our file now looks like this:

```
1 Page(1)
2 cls "Our
3 swap wait(50) ypos(30) c"Summer
4 swap wait(50) ypos(60) r"Vacation
5 swap
6 wait(0) q
```

Now when we test our file, the strings appear in an instant. We may think our delays are a little too long. We can change this by changing the wait value in lines 3 and 4 (higher numbers wait longer) or we can change the speed of the entire program by using the speed command. Speed ranges from (1- very fast) to 255 (very slow) with the default value being 64. If we edit line 1 to say: Speed(40) page(1), we now can do a test and see things moving a little quicker. You will also note the final screen does not stay on as long and the wait(0) command is also affected by the speed.

Lets enhance our program even further by increasing the size of the characters in our string. The size command can do this. A size of 0 is normal size, 1 is double wide characters, 2 is double height characters and 3 is double height and width. By editing line 1 to include a size(3) command we now see our screen very full!

We can have each string in a different font. The fonts command changes our fonts by font(x) where x runs from 0-7. By changing our program around to:

```
1 size(3) speed(40) page(1)
2 cls "Our
3 swap wait(50)
4 font(1) size(2) ypos(30) c"Summer
5 swap wait(50)
6 font(3) size(1) ypos(60) c"Vacation
7 swap
8 wait(0) q
```

You begin to see how diverse your text can be; but hold on, we have not yet begun to vary our text! We can also change the color by the color command (with a color of 1 to 3) and change what that color is on the screen by the pen command (with each color given a value as well as the background color). You may have only 4 colors on the screen at one time, so the pen command will change the color of anything previously put on the screen to that color hue. We can also change the style of the characters being printed by the style command. A value of 1 shadows each character, 2 outlines it and 3 causes a raised shadow characters. These styles can be further enchanced by the addition of italic (adder of 4 to style) and/or boldface (adder of 8 to style). The style command can be entered (as all command parameters can) as an expression (i.e. 2+4 yields italic outline). If we now get our program to look like this:

```
1 size(3) speed(40) page(1)
2 cls style(1) "Our
3 swap wait(50) font(1) size(2)
4 style(8) ypos(30) c"Summer
5 swap wait(50) font(3) size(1)
6 style(4) ypos(60) c"Vacation
7 swap
8 wait(0) q
```

Figure 4

You will find that not all styles look terrific with all fonts; in general, the bigger the size the better the effect.

Part II - 80 x 48 Bit Image Graphics

Bit image graphics are large pictures. They are generally used for logos and big pictures. In size(0) a graphic takes up one fourth of the screen (at size (3) it takes up all of the screen). The styles of italic and bold do affect graphics and the styles of outline and shadows have no affect. Graphics are generated in the form of GRAPH(n) where n ranges from 0 to 7. A typical graphic example might be:

```
1 page(1) cls c"LJK Enterprises, Inc.
2 pos(40,24) [to center on the screen
3 color(1) graph(0) [the LJK logo
4 font(1) style(12) color(2)
5 ypos(80) c"Presents
6 swap wait(0)
7 wait(0) q
```

Figure 5

Now if we were to change line 6 to read wipe(1) instead of swap, you will see the screen unfolding to you from the center out as an eye opening up. Wipe parameters are: (0) same as swap; (1) iris out from center; (2) come from left to right; (3) come from right to left; (4) come from top down; and (5) come from bottom up. Another thing to try would be to change the wait(0) in line 7 to read 'key' instead. What this will do is wait until a key is pressed before continuing. In this way, you can have the screen on as long as you want; allowing you to change cables or whatever and then pressing a key for the program to continue.

Part III - Block Animation

The third type of graphic generation in Movie Credits is block animation. That is groups of blocks of 8x8 bit images are manipulated and moved as a block. From a clean editor (use the NEW command), type the following:

```
1 [Two Cars Passing on a ship]
2 pos(0,80) pattern(20,1,104)
3 wait(0) q
```

Now when we test this program, we see we have drawn a brick road at the bottom of the screen. Patterns are blocks of **the same character.** Now lets expand our program by inserting three lines to generate the following:

```
1 [Two Cars Passing on a ship]
2 pos(0,80) pattern(20,1,104)
3 pos(0,74) define(0,3,1,98)
4 pos (136,74) define(1,3,1,101)
5 draw(0) draw(1) wait(100)
6 wait(0) q
```

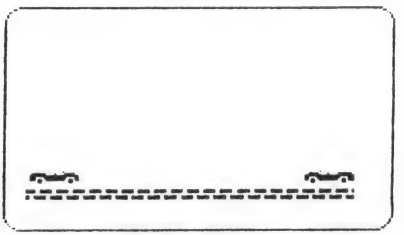


Figure 6

Now we have defined two character blocks, #0 which is 3 wide and 1 high starting with character #98. This means on the screen it will look like 98 99 100. We have positioned the cursor at 0,74 when we defined it. We have also defined block #1 at 136,74 to be a 3 high 1 wide block starting at character 101. The next line draws these blocks on the screen. We now test our program and see we have cars pointed in opposite directions on our brick road.

Before animating our characters, we need to introduce the simple loop construct of repeat(n) .... again. This will cause the area inside the repeat again segment to be executed n times. If we modify our program to now read:

```
 1 [Two Cars Passing on a ship]
 2 pos(0,80) pattern(20,1,104)
 3 pos(0,74) define(0,3,1,98)
 4 pos (136,74) define(1,3,1,101)
 5 draw(0) draw(1) wait(100)
 6 repeat(17) [do 17 times]
 7  right(0) left(1)
 8 again
 9 wait(0) q
```

We can now 'animate' our cars by moving them with the up, down, left and right commands (with cars we do not recommend up and down!). The repeat sequence will recreate the old silent films sight gag of cars coming head on through each other without accident (this was usually done with trains and a tunnel, but we didn't even need a tunnel!). We complete our program by erasing the objects to remove them from the screen. This is accomplished with:

```
 1 [Two Cars Passing on a ship]
 2 pos(0,80) pattern(20,1,104)
 3 pos(0,74) define(0,3,1,98)
 4 pos (136,74) define(1,3,1,101)
 5 draw(0) draw(1) wait(100)
 6 repeat(17) [do 17 times]
 7  right(0) left(1)
 8 again
 9 erase(0) erase(1)
10 wait(0) q
```

To further enhance our animation we need to introduce variables and their usage. We have 16 signed integer variables allowed in the program. They range from -32767 to +32767 and are accessed by V(n) where n is a number (or an

expression in the range of 0-15. We can assign variables by V(n)=a where a is an arithmetic expression; or we can increment or decrement a variable by V(n)+=a (This is comparable to V=V+a in basic).

More advanced graphics blocks animate by not only moving but also by changing their 'look' on the screen. For example we have defined in the block character set a number of helicopters. Each helicopter looks like the others with the exception of the rotor being in a different location. To demonstrate enter the following:

```
 1 [Helicopter Animation]
 2 pos(0,0) define(0,3,2,32) v(0)=32 v(1)=6
 3 draw(0) repeat(20)
 4  v(0)+=v(1) if v(0)=44 v(1)=-6
 5  if v(0)=32 v(1)=6
 6  char(0,v(0)) draw(0) wait(5)
 7 again
```

We have also introduced the if concept here. If the expression (for example [v(0)=44] in line 4) evaluated as true, the rest of the line will be executed (v(1)=-6) otherwise, it will be ignored.

When we test this program, we see a helicopter on the top left of the screen that seems to be hovering with its rotor going around. This is accomplished by changing the character of block 0 to correspond to the variable v(0). We have defined different helicopters (3x2 so each takes 6 block locations) at character locations 32, 38 and 44. We use V(1) as a direction variable, we either go up 6 or down 6.

Now we animate our helicopter by continuing its rotor motion but also moving the copter as well. We get our program to look like:

```
 1 [Helicopter Animation]
 2 pos(0,0) define(0,3,2,32) v(0)=32 v(1)=6
 3 draw(0) repeat(20)
 4 v(0)+=v(1) if v(0)=44 v(1)=-6
 5 if v(0)=32 v(1)=6
 6 char(0,v(0)) draw(0) wait(5)
 7 again
 8 repeat(10)
 9  v(0)+=v(1) if v(0)=44 v(1)=-6
10  if v(0)=32 v(1)=6
```

```
11  char(0,v(0)) right(0) again
12 repeat(9)
13  v(0)+=v(1) if v(0)=44 v(1)=-6
14  if v(0)=32 v(1)=6
15  char(0,v(0)) down(0) wait(5) again
16 repeat(9)
17  v(0)+=v(1) if v(0)=44 v(1)=-6
18  if v(0)=32 v(1)=6
19  char(0,v(0)) up(0) wait(5) again
```

You may notice we placed a wait(5) when going up and down and not when going right. This is because some things take longer than others, so we place in compensation to keep things smooth.

Now let's consider the following program:

```
 1 [Pattern Blocking]
 2 v(0)=105 page(1) gap(4) cls
 3 color(1) font(4) size(2)
 4 ypos(30) style(8) c"MOVIES
 5 size(0) color(3)
 6 pos(20,24) box(120,30)
 7 style(0) repeat(23)
 8  pos(0,12) pattern(20,1,V(0))
 9  pos(152,12) pattern(1,6,V(0))
10  pos(0,12) pattern(1,6,V(0))
11  pos(0,60) pattern(20,1,V(0))
12  swap v(0)+=1 wait(100) again
13 font(0) color(2) gap(0)
14 ypos(88) c"0C3 MCMXCIC2 By LJK, Inc.
15 swap key q
```

This program surrounds the string MOVIES (in Large Roman Bold font) by a continually changing pattern of characters to create a "box" of emphasis around the word MOVIES. The last group of characters in the predefined block set given with the program are designed for just that purpose. We have also added the command Gap to the MOVIES string. You will note this string is printed with a much wider space between the characters as a result of the GAP command. The final new addition is the copyright notice at the bottom of the screen. Generating the ` character (CTRL-. from the keyboard) in all LJK characters sets generates the copyright symbol. We have also varied the color of the year to be different that that of the rest of the string by generating

inverse characters (the inverse mode is set by hitting the Atari key and reset by again hitting the Atari key). We generated an inverse C to tell us to change color; an inverse 3 to set that color to 3 and later restored the color to 2 after printing the year. You can likewise change the font during a string print with an inverse F. Boldface, Italic, Outline, Shadow, Xsize, Ysize, going Up and going down can also be toggles by inverse BIOSXYUD respectively. With these commands, you do not need a following number as none is required.

Part IV - Graphic Primitives

Graphic primitives are two dimensional geometric shapes. They can be combined to create pictures to enhance your output. Typical primitives are things like lines, boxes, circles and arcs. Let's consider the example of a man erecting a cross in the middle of the screen. You will find this requires not only the use of primitives; but also, the extensive use of blocks. Our example:

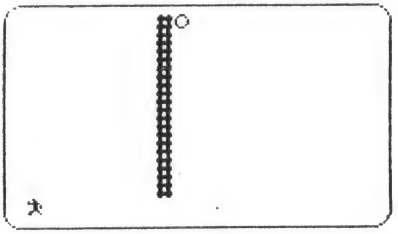


Figure 7

```
 1 [Graphics Primitives Example]
 2 v(0)=88 page(1) cls
 3 color(1) xpos(64) pattern (1,11,127) [ladder
 4 swap color(3) page(0) pos(0,88) define(0,1,1,6)
 5 draw(0) repeat(8) right(0) wait(10) again
 6 char(0,7) color(1)
 7 repeat(2) up(0) pos(64,v(0))
 8  block(1,1,127) wait(10) v(0)+=-8 again
 9 pos(76,0) circle(3)
10 repeat(11) down(0) pos(64,v(0))
11  block (1,1,127) wait(10) v(0)+=8 again
```

```
12 char(0,8) repeat(8)
13  left(0) wait(10) again
14 pos(64,0) pattern(1,11,0)
15 pos(8,88) line(64,-84)
16 pos(79,5) line(0,54)
17 char(0,6) draw(0) wait(50)
18 color(2) pos(76,60) fill(6,40)
19 pos(70,72) fill(18,6)
20 window(14,32,4,12)
21 repeat(4) sup(20) again
22 window(0,40,0,12) [full screen
23 color(1) pos(79,31) gmode(2) [xor mode
24 repeat(27) plot rpos(0,-1) wait(5) again
25 pos(0,88) line(64,-84) wait(20)
26 pos(76,0) circle(3) plot
27 gmode(0) wait(100) erase(0)
28 wait(100) swap page(1)
29 ypos(2) color(3) font(2) c"The F1Crusades
30 swap color(1) ypos(86) font(0)
31 c"A Pictorial History
32 swap wait(0)
33 wait(0) q
```

The program consists of a patterned ladder, a block man but then involves primitives by creating a pulley (circle) and the rope (lines). The cross is added by doing fills.

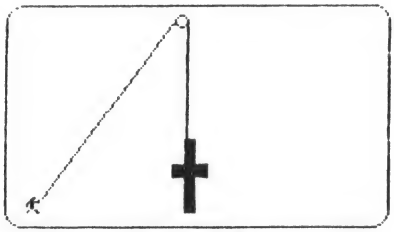


Figure 8

We were able to 'lift' the cross by creating a smaller window and scrolling that window up (lines 20-22). By setting the gmode to 2 in line 23 we set **exclusive or** mode for use to erase existing graphics. We then slowly erase the rope holding up the cross and finally remove our pulley and ladder and man. By the time the program is completed,

you will see we have combined almost all of our techniques to create a rather complex video affect.

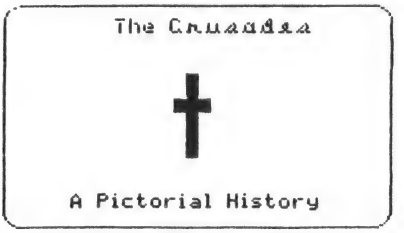


Figure 9

Figure 9 now shows us the final constructed cross centering our title. And as they say in the film industry, Voila, the rest is history....

Part V - Screen Builder

An alternative form of entry is the screen builder. The screen builder allows for what you see is what you get type editing. The limitation of the screen builder is that once a screen is sent back to the main program it can no longer be edited by the screen builder. To reedit a screen, you must either edit it under the normal Movie Credits edit commands or rebuild it completely in the screen builder. We are going to build a wedding flyer similar to the one shown in figure 10:

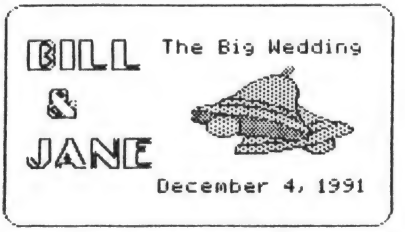


Figure 10

Starting with a clean editor (by typing new and verifying) we add the line of **page(1) cls.** This sets us up to draw on page one while viewing page zero. The cls gives us a clear screen to start with. We then proceed to invoke the screen builder by typing **brun sbld.com**:

We first select a text entry by either pressing [Return] (since we are positioned at the text option) or pressing a T. We are then prompted on the top line for the text. We enter **BILL** and press return. We are now looking at a box in the middle of the screen. This box represents our entry. We can examine the commands at our disposal by typing the ? for help.

We wish BILL to be in the big Broadway font (font #7). We select this font by pressing G to get to the goods menu and F to select font. When prompted on the top line for the font, we enter 7. You will notice that our box has become more than twice as big as it was. We can move our box around by using the non-controlled arrow keys (i.e. -, =, + and *) to move in eight positions around the screen or the arrow keys in conjunction with the control key for fine motion (1 location at a time). Try moving our box around with these keys at this time.

We can also move the box along the x-axis with the positional keys of 1 to move to the left edge; 5 to center the box on the x-axis and 0 to place the box on the right edge of the screen. We can likewise move the box along the y-axis with the positional keys of , to move to the top of the screen; . to center the box on the y-axis and / to place the box to the bottom of the screen. Try this motion now. You will notice placing the box on the top edge has some of it hidden underneath our menu bar. This is ok. When the screen will be shown, the menu bar will not be there.

We now wish to truly position our box of "BILL" on the left edge at a y position of about 20. When we are there, we press [Return] to accept the entry. We again go back to our menu for the next entry. We now enter & and go into our next entry. We have decided now, that the name strings will look better in outline font. We simply press S for style and then O for outline. You will notice that our previous string "BILL" does not change to this style, but we can change that during editing. We position at the left edge (with the 1 command) and move in to center under our previous entry by pressing the * a couple of times. We press [Return] to enter this option. We now go back and add **JANE** placing it on the left edge equidistant from the & as the BILL string only below the *. You will notice this string comes up in the outline mode already.

We now wish to add an graph to our screen. We press G from the menu and are prompted for the graph to which we enter 3.

Our box is now 80 by 48. We wish this to be in color 1 so we press **G** to get to the goods menu and **C** for color. We select 1. You will notice the goods menu no longer has options for font and gap. They are reserved for text operations. Likewise the style menu will only have options for bold and italic. We position our color 1 graphic box on the right edge with the **O** command enter it there. We now place in normal font 0, color 2 the strings "The Big Wedding" above the graphic and "December 4, 1991" Centered below the graphic.

We have now entered all of our information. We know we have to edit the BILL string to be in outline mode. We press [Esc] to exit add mode and come into edit mode. We now wait as the entire screen is redrawn. We now have a box cursor surrounding the current entry. We can again press ? for help. You will notice the commands have changed as we are in edit mode. We can move our cursor box by pressing [Esc] to go to the first entry or by using the arrows (with or without the control keys) to go from entry to entry. Try this now. We wish to edit our first entry so we get there by pressing [Esc]. We edit the entry by pressing [Return]. The top line is prompting us for the text and having us edit the String BILL. Since the text is ok, we just press [Return]. It seems as if nothing has happened, but you will notice the top menu bar now is for the adding and editing of our entries. We press **S** to change the style and **O** to get it to outline. Our box is now bigger (because of the outline) but the string has not changed. The box will move if you reposition the string but the text will remain until you complete the editing of that entry by pressing return. We may also wish to reposition this entry a little higher for better space balancing, we do this by moving it up with a -. Pressing [Return] ends our edit of this entry and the screen is redrawn. We now wish to move our JANE entry down a little to balance the screen a little vertically. This is done by moving to the JANE entry with the arrow keys, selecting it with [Return], accepting the text as is with [Return], and repositioning it with the arrows and ending the edit with [Return].

At this point if we felt we had forgotten something in our screen we could press **A** to return to add mode to add more. We like the screen the way it is and now wish to send it to the program. We can now get to the exit portion of a screen. We either press Save or **Q** to get there. Press **Q** to quit and our asked to verify this operation. Leaving the screen builder at this point means we can no longer edit

this particular screen with the screen builder unless we recreate it from scratch. We answer with a **Y** and now see the wipe menu. We can add a line of commands to act as a screen transition here if we choose; we can exit without adding a wipe at the end by pressing [Esc].

A typical transition line might be **wipe(2)cls wait(0);** where the wipe(2) brings the screen to view, the cls clears our screen and prepares for the next screen, and the wait(0) is a delay to keep the old screen in view for a long time.

To create a typical wipe line, we press **L** from the wipe menu for our left wipe, then the forward arrow key to get to the feature menu, a **C** generates a cls and hitting **W** for a wait prompts us for the delay time. Pressing [Return] yields us the default value 0.

If at some time in this process we have made a mistake or wish to change how the line looks, we can go over to the exit menu and press **Redo** to clear the line and start over. When we have completed our line, we can exit by pressing [Esc] or by hitting **Quit** from the exit menu. To not enter a transition line, just exit when there is nothing on the line.

That's all there is to it. We exit with the **Quit** and find ourselves back in the Movie Credits command mode. We can now test our new screen with the **TEST** command and we see our wedding flyer wiped in from the left.

Part VI - Transferring To Tape

When the files have been loaded into the machine, we type **run [return].** We are now prompted with **Head, Trail, File** or **Quit?** Typing the first letter of the word will elicit the appropriate action. We now start our tape in the record mode and press **H** for the header section of our tape. When this completes, we simply stop the recording and go on to record our program. In this case, since we have no program we assume that we did that independently and have now completed it. We now wish to add the trailer section to our video tape. This is accomplished by pressing the **T** and immediately recording the trailer. When this operation is complete we now have a tape with a header section, a body (from your recording) and a trailer. If we were doing more than one tape, we could have used the **F** selection when we were changing files.

The Editor

LJK Movie Credits is a programming language specifically designed for manipulating the four image types. The program executes commands generated from the line oriented text editor that is part of Movie Credits. The Editor is the area where you write and edit programs, and send programs to and from your disk. It is in the Editor that additions, deletions, and corrections are made.

The Editor is a line editor; that is, you work with one line at a time. There are several commands to help you work with and move about on the line you are editing; these are listed in below. The form of the commands given to the editor are: command [parameter] where parameter is in [] to indicate that it is not always required. Parameters can be things like line numbers, filenames, search strings and the like. Line number ranges are specified like this:

```
COMMAND          All lines, starting at line 1
COMMAND X        Line X only
COMMAND X,       Line X to the end of the file
COMMAND ,Y       Line 1 to line Y
COMMAND X,Y      Line X to line Y, inclusive
COMMAND X-       Line X to line X+19
```

Strings are optionally delimited. That is they normally are delimited by commas(,). If you wish a comma in your string, you need to delimit it by either a" or a '. This is also the case if the string starts with a space or a number (the program will think it is a line number). Examples of string delimiting are:

```
LOAD file.src       String is file.src
FIND   mylife       String is mylife
F long, long str    String is long (, delim)
F "long, long       String is long, long
R 12,35 " me" mine  search string is " me",
                    rep string " mine"
```

Input format is free form. That is spaces can be anywhere between the commands and the parameters. Spaces can not be embedded in the commands themselves.

When listing, finding or searching, you can pause the listing with the space bar and abort the operation with the [BRK] key.

Editing Commands

A or **ADD** -- Add keyboard input lines to the end of source text in memory. The system automatically prompts with the next line number. Input is accepted until a null line is entered (i.e. hitting esc at the start of the line).

```
Examples: Add
          A
```

COPY LN01,LN02;LN03 -- Copy lines LN01 through LN02 to before LN03. The original text remains as is. The line numbers will change after the copy). The copy command cannot be abbreviated.

```
Examples:  copy 5,10;15
           copy 20,25;30
```

DEL LN01 [,LN02] -- Deletes either a single line or all lines in a range. There is no prompting for verification on Del and the process is irrevocable.

```
Examples:  del 5
           del 10,15
           DEL 20,
```

E or **EDIT** [LN01] [,LN02] -- Edit one or more lines of existing source text. Each line is displayed with the cursor in the first position; ready for you to re-type all or part of it. Within a range, to skip editing a particular line, just press **RETURN**; the system continues with the next line. If LN01 is omitted, line one is assumed. If you cancel a line with **[SHIFT][DEL]**, the original line is brought up for re-editing. If you break (with [BRK]), the original line is left in the file as it was.

```
Examples:  e 5
           EDIT 20,25
           E 30,
```

F or **FIND** [LN01] [,LN02] STRING1 -- Find and list on the video screen all occurrences of STRING1 within a range of linenumbers. Each find will list that line on the screen. The find can be paused with the space bar and aborted with

[BRK]. The search is case insensitive (i.e. junk is the same as JUNK).

```
Examples:  f head
           F 5,10 "35
           find 15,20 " Space lead
```

HELP -- Displays the two sets of commands. The first list is the editor commands and the second set is the program commands.

I or **INS** [LNO1] -- Insert keyboard input lines before LNO1. system automatically prompts with the line number, which is incremented after each **RETURN**. Input is accepted until **RETURN** is entered as the first character of a line. Your first line becomes the "new" LNO1; if LNO1 is omitted, line 1 is assumed. If you cancel with **[BRK]**, then that line is not entered and the Insert is aborted.

```
Examples:  i 5
           I 10
           ins 20
```

L or **LIST** [LNO1] [,LNO2] -- List the text from LNO1 through LNO2. If no line numbers are given, all the text is listed. If only one number (and no comma) is entered, then only that line is listed. The listing can be paused with the space bar and aborted with [BRK].

```
Examples:  l
           list 5,10
           L 15-
           list ,30
```

M or **MOVE** LNO1,LNO2;LNO3 -- Move text from LNO1 through LNO2 and insert before LNO3. The text is then deleted from its original location; line 'LNO1' becomes the "new" line 'LNO3'.

```
Examples:  m 10,15;20
           m 25,30;35
           move 40-;10
```

NEW clears the text buffer to empty. You will be asked if you really want to Verify (Y/N)? Reply **Y** or **N** to the query. If you respond with the Y, your file in memory is no longer accessible. Use **ADD** after **NEW** to enter a new source program.

R or **REP [LNO1] [,LNO2] STRING1,STRING2** -- Change all occurrences of STRING1 to STRING2 in each line (within the specified range) that contains STRING1.

```
Examples:  r PRM,PARMS
           rep 5,10 " lo-" LOW-
           r ,15 "LABEL1"LABEL2"
           r 20, 'DINE'DONE'
```

RUN -- Switch to the video tape mode. The screen will now prompt with Head, Trail, File or Quit? Control will return to the MC: prompt (and the editor) on pressing Q.

S or **SRCH** [LNO1] [,LNO2] STRING1,STRING2 -- Selective change; you are prompted as each occurrence of STRING1 is found. To change STRING1 to STRING2, type a **Y** when the line is shown and to not make the change, type an **N**.

```
Examples:  s THIS,THAT
           s 5,10  ALPHA,Alpha
           SRCH 20- "LARGE"BIG"
```

STAT -- Show status of file. First the drives on line are listed. Each drive will have its density listed (S for single, D for double) and the currently selected drive will be in inverse video. Then the file is shown in its size and the currently remaining free space. Each value is show first in decimal and then in hex (preceded by a $).

TEST -- Run the video program with control returning to the editor. This is the way to test routines and then re-edit them. The mode (head or trail) will be the last mode accessed. When booted, Movie Credits defaults to the trail mode.

File Commands

APND filename -- Takes a file from disk and appends it to the end of the file currently in the Editor. If the file will not fit in memory, an error will be generated and the file will not be loaded.

```
Examples:  Apnd fbi.lib
           Apnd nxfile.txt
```

BLOAD filename -- Loads a binary file image into memory at its default location. This command is used for changing fonts, graphics or blocks.

```
Examples:  Bload mc.fnt
           BLOAD mygr.gr
```

BRUN filename -- Executes binary program file that is an adjunct program to Movie Credits. Examples might be a font builder, a screen builder, a character editor or a graphic editor.

```
Examples:  Brun gedit.com
           Brun fbild.com
```

DEV d [D][S] -- Selects current drive to be d with a range of 1 to 8. If an optional S is afterwards, the drive will be set to single density; if a D is afterwards, the drive will be set to double density. If there is no character afterwards, the density will remain as currently set.

```
Examples:  DEV 2
           dev 1d
```

DIR [filespec] -- Gives a directory of all files meeting the filespec on the current drive. If no filespec is given, it is the same as *.* and all files will be listed.

```
Examples:  Dir
           Dir *.txt
           DIR demo?.txt
```

ERASE filespec -- Erase all files meeting the requirements of filespec. Files that are not locked will be removed from the disk. This is a non-recoverable delete.

Examples: Erase myfile
Erase file?.*
Erase *.*

FMT String -- Format blank disk in the current drive. The disk will be formatted under either single or double density according to its current setting. The volume id string that will be shown during a directory can be up to 15 characters long.

Examples: fmt Inverse Title
fmt " My Files #3

LOAD filename -- Loads a file from disk into the Editor. Caution: Loading a file erases whatever had been in the Editor prior to the Load command. The filename can be typed in upper or lower case.

Example: load myfile

QUIT -- exit program by rebooting the entire system.

SAVE filename -- Saves the file currently in the Editor to disk under the name filename. Filenames have an 8 character primary name and a 3 character secondary name, the names are separated by the ".". The first character of all filenames **must** be alphabetic. If filename exists on the disk, it will be overwritten unless it is locked and then an error will occur.

Example: Save new.txt

Movie Credits Commands

Movie Credits commands are made up of the command and possibly a set of parameters associated with that command. Most commands can be placed after another command on the same line. These commands are not case sensitive (i.e., commands can be entered in either upper or lower case) and their format is free form (spaces can be added for indentations and clarity); spaces embedded within commands are not allowed.

Numerical Parameters

Numerical entries of parameters may be mathematical expressions of signed integer numbers. These numbers range from +-32767. The default input mode of the numbers is decimal; but this can be overridden to hexadecimal input (hex) by preceding the number with the dollar sign ($). The priority of operators is listed below. This priority can be superseded by the use of parenthesis around that portion of the expression. The priority list:

1. unary (-) and parenthesis (())
   (highest priority).
2. Multiply (*), divide (/) and modulo (\).
3. Add (+) and subtract (-).
4. Booleans (<, <=, =, <>, >=, >)
   (lowest priority).

Booleans will yield a 1 if true and a 0 if false.

Numeric parameters should be parenthesised for easiest understanding. Movie Credits will add parenthesis for you and try to place zero value numbers for missing results.

String Parameters

Strings are the group of characters that follow the command until the end of the line (cr) is reached. Mostly they are just printable characters that you wish to see on the screen. Any characters that are in inverse video (toggled on and off by the Atari key) are not printing characters, but may be used to generate further enhancements to the text. The inverse [B] will toggle bold faced type on and off; likewise and inverse [I] will toggle italic; [O] outline; [S] shadow; [X] character width (single or double);

[Y] character height (single or double); [F][x] will change the font where x should be an inverse number from 0 to 7; and, finally, [C][x] to change the color where x is an inverse number in the range of 1-3.

SCREEN COMMANDS

" string - Prints string at current cursor position.

C" string - Prints string at current y location centered on the x-axis.

R" string - Prints string at current y location so that the right edge of the string is at the right margin of the screen.

Bell - sound the audio bell as a cue to the user, this audio information will not be generated to the video tape.

CLS - clears the current window to the background color.

CR - generates a carriage returns and places you on the next line. This will advance you either 8 or 16 "y" positions depending on the current font selected.

KEY - waits for a key to be struck before continuing execution.

SDN (expr) - scroll screen down 1 text line (8 or 16 y) and wait for time expr.

SUP (expr) - scroll screen up 1 text line 8 or 16 y) an wait for time expr.

[- ignore the rest of the line as a comment.

LOOP CONTROLS

AGAIN - signals the end of the loop construct.

HEAD - will execute the rest of the line if we are in the head mode, otherwise it will ignore it.

IF (expr) - will execute the rest of the line if the expression expr evaluates to a non-zero, else the rest of the line is ignored.

REPEAT (expr) - starts execution of a repeat - again loop which will be executed expr times. Loops can be nested up to 10 deep.

TRAIL - will execute the rest of the line if we are in the trail mode; otherwise the remainder of the line will be ignored.

V(expr)=val - assigns to the variable expr (where expr has a range of 0-15) the value val; val is a signed integer expression with a range of +-32767.

V(expr)+=val - adds the value of val to the current variable expr and stores the result in variable expr.

GRAPHIC PRIMITIVES

ARC (start,end,rad) - generate a clockwise arc starting at angle start and ending at angle end with a radius of rad. The angles start and end are in degrees and end can be less than start.

BARC (start,end,rad) - generate a counterclockwise arc starting at angle start and ending at angle end with a radius of rad. The angles start and end are in degrees and end can be less than start.

BLOCK (hor,ver,char) - generate a block of characters hor wide by ver high starting at character char at the current cursor position.

BOX (x,y) - draw a line box relative to the current location with a width of x and a height of y.

CFILL (expr) - draw a filled circle with a radius of expr at the current location. The current location will be the top of the circle.

CHAR (num,char) - change the predefined block character in block num to the character char.

CIRCLE (expr) - draw a circle with a radius of expr at the current location. The current location will be the top of the circle. This is the equivalent of arcr (0,360,expr).

DEFINE (num,hor,ver,char) - define block number num (range 0-7) to have the conditions of hor width, ver height and start at character char. The block will be located at the current cursor position.

DOWN (expr) - move block number expr down 1 location. This will erase and move for animation.

DRAW (expr) - draw block number expr at its current location.

ERASE (expr) - erase block number expr from its location. This will replace the block location with background color.

FILL (x,y) - fill in a box relative to the current cursor location x wide and y high.

GRAPH (expr) - draw graphics image (80x48) expr at current location. Expr has a range of 0-7.

LEFT (expr) - move block number expr left 1 position. This will erase and move for animation.

LINE (x,y) - draw a line with ending with a relative horizontal position of x and a relative vertical position of y.

PATTERN (hor,ver,char) - draw a pattern of repeating character char that will be hor wide and ver high at the current location.

PLOT - plots a dot at the current cursor location.

POS (x,y) - position the cursor at horizontal location x and vertical location y.

RIGHT (expr) - move block number expr right 1 position. This will erase and move for animation.

RPOS (x,y) - relatively move the cursor to the horizontal offset x and vertical offset y.

UP (expr) - move block number expr up 1 location. This will erase and move for animation.

XPOS (expr) - set the horizontal location to expr.

YPOS (expr) - set the vertical location to expr.

MODES AND SUCH

!0(f,m,s,x,y) - Internal command where f represents the font plus the gap times 8; m represents the mode with the size being the most significant two bits followed by the graphics mode for the next two bits and the color being the least significant two bits (i.e. a size of 1 [double x] would add $40 [64 decimal] to the mode byte; s is the style will all the additions as normal based on clipping, font style, bold and italic; x represents the x position [0-159]; and y represents the y position [0-95].

!1(m,s,x,y). - Internal command where m represents the mode with the size being the most significant two bits followed by the graphics mode for the next two bits and the color being the least significant two bits (i.e. a size of 1 [double x] would add $40 [64 decimal] to the mode byte; s is the style will all the additions as normal based on clipping, font style, bold and italic; x represents the x position [0-159]; and y represents the y position [0-95].

COLOR (expr) - set the current color to be expr. Expr runs from 1-3. 0 will generate all background.

DISPLAY (expr) - set to display page 0 or page 1 of the two video pages.

FONT (expr) - select the font expr. Currently fonts are either 8 or 16 bits high. All are proportional in spacing. There are eight fonts numbered from 0 to 7.

GAP (expr) - sets dot gap between characters; expr can be 0-15.

GMODE (expr) - set the graphics mode. Graphics modes are: (0) put mode; (1) or mode; (2) xor mode.

PAGE (expr) - sets page to do writing to. This is a zero or a 1. This will not change the current display page.

PEN (a,b,c,d) - sets current palate for the 3 operating colors plus background. The values a, b, c and d are defined under standard Atari color schemes. d is the background color. Each color is made up of two parts, a hue and an intensity. It is usually better to give the pen colors in hex as this differentiates them by digit. The hex digits for color are listed below:

Hex	Color	Hex	Color
0	Gray	8	Blue
1	Gold	9	Light-Blue
2	Orange	A	Turquoise
3	Red-Orange	B	Green-Blue
4	Pink	C	Green
5	Purple	D	Yellow-Green
6	Purple-Blue	E	Orange-Green
7	Blue	F	Light-Orange

The intensity goes from off(0) to bright (E) by even values (i.e., 2,4,6). A pencolor is made up of a color nybble and an intensity nybble. For example, a medium bright blue would be $78. Default pen colors are: $28, $ca, $94, $00.

RESET - set all modes back to their default values. This will not affect the variables or the loops. Specifically, gmode, gap, size, style, font, page and display are all set to 0; color to 3; the pen is set to the default colors, the speed is set to 64 and the window to (0,40,0,12).

SIZE (expr) - sets x and y size for character, block and graphic prints. 0=normal, 1=wide, 2=tall and 3=tall and wide.

SPEED (expr) - sets speed to expr. Expr can be 0 (256) - 255. Where 1 is the fastest speed. The speed defaults to 64.

STYLE (expr) - set the font style and orientation. All parameters are additive by groups. Styles are: 0-normal; 1-shadow; 2-outline and 3-raised shadow. Additives are:

$04	Italic
$08	Bold
$00	Text from left to right (->)
$10	Text from right to left (<-)
$20	Text from bottom to top (↑)
$30	Text from top to bottom (↓)
$40	Text from top/left to bottom/right (\)
$50	Text from bottom/right to top/left (\)
$60	Text from bottom/left to top/right (/)
$70	Text from top/right to bottom/left (/)

The styles are for characters only; while the enhancements work on graphics and block characters too. This is an additive expression by bits. For example, to select italic

bold we would say style(4+8) or style(12). Adding 128 ($80) to the value will cause the screen to not scroll.

SWAP - swaps the current written video screen over to the other one. This is highly useful for writing on page while viewing the other. See also wipe.

WAIT (expr) - delay for time expr; expr can be 1 (short) to 255 (long).

WIPE (expr) - swap the screen from the current writing display page to the other display page by doing one of 6 different wipes. 0=swap, 1=iris from center out, 2 through 5 represent left, right, down and up respectively.

WINDOW (a,b,c,d) - sets left edge, right edge, top margin and bottom margin respectively. These are on byte boundaries (i.e. bottom of 12 is 12*8 or 96). Default values (maxes) are 0,40,0,12.

Q - quit execution at this point and return control to either the editor (if we got here by test) or the head/trail/file prompt.

Color Control Generator

Purpose: To be able to adjust pen color hues and intensities on a what you see is what you get basis.

Technical: Each color is made up of two components, the hue (or color) and the intensity (brightness). The color control generator allows individual adjustments on each component for each of the three colors and the background color.

Execution: The program is run by typing **brun pen.com** from the Movie Credits command mode. Colors 0-3 will be printed along the left edge (0 represents the background color) and there will be an H and an I box for each color. You will also be shown the current color value in hex on the right side of the screen. Each color will be shown in its current color value with the exception of the background color which will be shown in color 3 (if we showed it in its color you wouldn't see it!).

The commands available to you are:

?	Display the help screen
<arrows>	motion from one color to the next
Esc	Abort, no pen command
Return	Exit accepting current values
R	Reset standard color values
>	Increase current hue/intensity
<	Decrease current hue/intensity

The program is exited by pressing [Esc] (no pen created) or [Return] which will create a pen command at the current setting of the color registers.

Screen Builder

Purpose: To more easily generate static commands than can be achieved using the text editor. The screen creates a "What You See Is What You Get" editing environment.

Limitations: Once a screen is sent back to the main program it can no longer be edited by the screen builder. To reedit a screen, you must either edit it under the normal Movie Credits edit commands or rebuild it completely in the screen builder.

Execution: The program is executed by typing **brun sbld.com** from the Movie Credits command mode. The program is pull-down menu driven. When the cursor is highlighting a menu selection, you may select that option by pressi g [Return], you may select another option from that menu (or the option you are on) by pressing the first letter of the options name. You may move the menu cursor up and down with the arrow keys (you do not have to press [CTRL] with the arrows), you may move from menu to menu with the forward and back arrows, or you can abort the menu operation with the [Esc] key.

The bottom status lines will tell you the number of entries. It will also show you the font, gap and color, the highlights of bold, italics, clip, the main style. The graphics mode will show up as a single character (a heart for put mode) and the orientation will show up as a graphic character. The x and y locations will also be shown. There are a maximum of 64 entries per screen with sixteen (16) of them containing strings (Text selection).

Upon entering the screen builder you will be at the Graphics Menu. The Graphics Menu contains two different menus. Graph will generate items to the screen and Mode will perform housekeeping operations. If you select from the Mode Menu, New will ask to verify before erasing. Quit and Save will transmit the screen already created to the main Movie Credits program without first going through edit mode. You will first be asked to verify with a Y or an N.

When pressing a selection from the Graph Menu, you will be asked to enter the input for that particular command (text asks for text, graph asks for the graphic #, block and pattern ask for the x, y and the character). You now are in add mode with the following commands:

-=+*	Move by 8 locations in direction
<arrows>	Move by 1 location in direction
1	Move to left edge (if orientation=0)
5	Center on x-axis
0	Move to the right on the x-axis
,	Move to top (if orientation=0)
.	Center on y-axis
/	Bottom basis y-axis
?	Display the help screen
Esc	Reenter Data
SODG	Select Menu Option
Return	Accept entry and location

When completing an entry (with [Return]) you will then be at the Graph Menu to add the next entry. You continue in this fashion until you press [Esc] at the Graph Menu which will place you into **edit** mode. The screen will be entirely redrawn (this can take a little time) and you will be boxed around the current entry.

You now may move about to different entries and edit them to the locations you desire. Your edit mode commands are:

<arrows>	Move from entry to entry
?	Display the help screen
Esc	Select the 1st entry
Return	Edit the current entry
S	Send this screen to the program and start new screen
E	Erase this entry
I	Insert Entry before this one
A	Return to adding more entries
Q	Quit editing and exit to program

Pressing [Return] to edit an entry will go through the information and allow you to replace the entry until you press return to accept it. Then the screen will be printed again and you will be returned to the edit mode.

Pressing **S** to save or **Q** to quit will transmit the screen to the main Movie Credits program. You will first be asked to verify the save with a Y or an N. Proceeding will then ask prompt you with the Exit Menu where you can create a transition line between different screens. As you progress, the line being created is shown near the bottom of the screen.

The Wipe options allow for creating one of the various wipes to bring the screen into view; under features, the options of cls, key, wait, perform just as **cls** will perform a clear screen, **wait** will prompt you for the wait time and perform a delay, **bell** will ring the bell, and **key** will pause until a key is struck. The exit option allows for a **redo** which will clear the existing line and start over, and quit will exit with the line at its current state (you can also quit by pressing [Esc].

Print Shop Graph Builder

Purpose: To generate new bit image 80x48 graphic objects for Music Credits by taking them from artwork generated by Print Shop or Print Shop companion.

Print Shop graphics are stored as 88x52 so they will be clipped by four columns on each side and by two rows on top and two rows on the bottom. Care should be taken to see that these areas do not contain significant graphic data.

Execution: The program is executed by typing brun psgr.com from the Movie Credits command mode. The program identification will come up and you will be asked the print shop drive. This is when you should change diskettes if disk swapping is required. To answer the drive command, do it as you would under the device command in the main Movie Credits program. If your drive is currently configured for double density, be sure to add an 'S' to the command to get the drive to single density.

A directory of the Print shop disk will be shown. This may encompass more than one screen. The directory can be paused by hitting the space bar and continued by hitting another key (or the spacebar again for line at a time movement). You will be prompted to enter each graphic in turn. If you need another directory, enter a null filename (just press return) for a graphic and the directory will be reshown.

When all graphics have been loaded, you will be prompted for the save drive. This again would be the time to swap diskettes if that is required. You may also change density at this time. You will then be asked the file name to save under. Entering the name of MC.GR and saving to a bootable Movie credits disk will have that be the graphic file loaded when the program is started. Other graphic files can be accessed by the bload command. If no name is entered, the file will not be saved to disk (upon returning to the program, you will still have these graphics in memory).

If at any time you hit the break key or encounter a physical error in accessing a disk or complete execution of the program, you will be prompted to 'Again (Y/N)? hitting Y will restart the program; hitting N will return you to the command mode of Movie Credits.

Technical: The 8 graphic images are stored as $200 byte long images starting at $4f00 (even though the length of each is $1e0 bytes). The entire graphic file saved will then be $fe0 bytes long. You could build the file from another source; just ensure that the file is binary starting at $4f00.

Graphics Builder

Purpose: To generate new graphic files by mix and match of already generated graphic files. This can be a combination of the default icons file and files generated from print shop icons using the Print Shop Graph builder.

Warning: The Graphics builder uses the text area as a buffer; so any program in memory will be lost upon entering the program. All files should be saved beforehand.

Execution: The program is executed by typing **brun gbldr.com** from the Movie Credits command mode. The program identification will come up and you will be asked the load drive. This is when you should change diskettes if disk swapping is required. To answer the drive command, do it as you would under the device command in the main Movie Credits program.

You will then be shown eight graphic icons on the screen. This screen represents your buffer of icons to be taken from (at this point your graphics builder buffer will also be the same). You will be prompted on the top line with **Graphic 0 (0-7) DFSVQ:** This means you are being asked to erect graphic number 0 in your buffer; the commands available to you are:

0-7	Select graphic from the viewing buffer.
D	Change the file device and/or density
F	Load a new file into the viewing buffer
S	Skip this graphic and keep the old one
V	View the builder buffer
Q	Quit this building as we have all we want

Selecting a numbered graph will place it in the buffer at location number 0. You will then repeat the process for graphic 1 through graphic 7.

When all eight graphics have been completed or you exited prematurely (with quit), you will be prompted for the save device. Then you will be asked the save file. Entering no filename will not save the current graphic to disk (but it will remain in memory). Selecting the name of **MC.GR** and saving it to a Movie Credits program disk will make that the bootup graphic file. By using the **mc.gr** designation as the saved file, you can create different program disks that boot with different sets of icons.

If at any time you hit the break key or encounter a physical error in accessing a disk or complete execution of the program, you will be prompted to 'Again (Y/N)? hitting Y will restart the program; hitting N will return you to the command mode of Movie Credits.

If you accidentally enter the wrong numbered graph, you may quit without entering a filename (no save) and rerun the program (you may wish to load **mc.gr** from the program disk as your start file as your old one is still in memory).

All directories from the graphics builder are done basis the filespec of "*.GR", so it is a good idea for the filename to follow that specification.

8 x 8 Font Adjuster

Purpose: To adjust 8 x 8 font files to prepare them for use in the font builder. This includes converting full 128 character fonts to the 96 character fonts used in Movie Credits, generating the copyright symbol and left justifying each character.

Technical: The normal Atari fonts contain 4 groups of characters (of 32 each); the punctuation, upper case, controls and lower case respectively. The Movie Credits font contains punctuation upper case and lower case in that order. Wide fonts do not work too well with the program as they are already bold faced to begin with and the bold option will be quite muddy (you are welcome to try them if you wish, however). Font files are saved as data files and not bit image binary files.

Execution: The program is executed by typing **brun fadj.com** from the Movie Credits command mode. The program identification will come up and you will be asked the font

drive. This is when you should change diskettes if disk swapping is required. To answer the drive command, do it as you would under the device command in the main Movie Credits program.

A directory of the font disk will be shown of files meeting the specification of "*.FNT". This may encompass more than one screen. The directory can be paused by hitting the space bar and continued by hitting another key (or the spacebar again for line at a time movement).

You will be prompted for a filename. The font will be loaded and adjusted for Movie Credits use. You then will be asked the save drive. This is the same device command as described earlier. The filename will be asked for then with the directory being shown under the filespec of "*.FN?". It is recommended that all Movie Credits editable fonts be saved as *.FNS for 8 x 8 fonts and *.FNL for 16 x 16 fonts.

If at any time you hit the break key or encounter a physical error in accessing a disk or complete execution of the program, you will be prompted to 'Again (Y/N)? hitting Y will restart the program; hitting N will return you to the command mode of Movie Credits.

Proportional Font Builder

Purpose: To generate new online proportional fonts from bit mapped editable fonts as a combination of 8x8 and 16x16 fonts.

Technical: The 8 online fonts are stored from $5f40 to $87ff in memory. The first portion of the file is tables to point to each character in each font. Space is tight. Using the same character by more than one font is exploited to the fullest in order to ensure more room. If you have no great unique character for a spot, use the standard one. Fonts go from character $20 to $7f inclusive. The program creates its own $20 and $7f character even though they exist in the edit font. Since character $7e is unavailable from the keyboard, it is recommended that the character be blanked to save space. Character $7b should be a copyright symbol in the set. Again, to save space, characters in the edit font should be left and top justified rather than in the center of the grid. Since the program creates bold faced by printing the character and moving over 1 pixel and

printing again, fonts that are inherently bold faced to begin with (as is the standard Atari font) do not make good Movie Credits fonts; the bold of them gets rather muddy.

Execution: The program is executed by typing brun fbuild.com from the Movie Credits command mode. The program identification will come up and you will be asked the font drive. This is when you should change diskettes if disk swapping is required. To answer the drive command, do it as you would under the device command in the main Movie Credits program.

A directory of the font disk will be shown of files with the filespec of "*.FN?". This may encompass more than one screen. The directory can be paused by hitting the space bar and continued by hitting another key (or the spacebar again for line at a time movement). You will be prompted to enter each font in turn. If you need another directory, enter a null filename (just press return) for a font and the directory will be reshown.

At name entry time, the files are not loaded but only checked to see that exist. The names are entered by editing the filenames of the default font files. When all eight names have been entered, the program will sequence through each file building fonts and showing the portion of memory used up. The program starts with 29 free pages. When all fonts have been loaded, you will be prompted for the save drive. This again would be the time to swap diskettes if that is required. You may also change density at this time. You will then be asked the file name to save under. Entering the name of MC.FNT and saving to a bootable Movie credits disk will have that be the font file loaded when the program is started. Other font files can be accessed by the bload command. If no name is entered, the file will not be saved to disk (upon returning to the program, you will still have these fonts in memory).

If at any time you hit the break key or encounter a physical error in accessing a disk or or run out of memory or complete execution of the program, you will be prompted to 'Again (Y/N)? hitting Y will restart the program; hitting N will return you to the command mode of Movie Credits.

If you run into memory problems due to fonts being longer than the original fonts a possible solution is to create some of your sets with 5 8 x 8 fonts and only 3 of the larger fonts.

Graphic/Character Editors

Purpose: To generate new bit image 80x48 graphic objects or to generate new 8x8 or 16x16 character sets. There are two programs involved here; Gedit.com which is the graphic editor and Cedit.com which is the font editor that switches between 8x8 and 16x16 operation.

Execution: The program is executed by typing **brun gedit.com** or **brun cedit.com** from the Movie Credits command mode. The command sets for the programs are (please note: commands that are only used for characters are preceded with an asterisk (*)):

U	Move cursor up and to the left (\)
I	Move cursor up (↑)
O	Move cursor up and to the right (/)
J	Move cursor to the left (<-)
K	Move cursor to the right (->)
N	Move cursor down and to the left (/)
M	Move cursor down (↓)
,	Move cursor down and to the right (\)

Note: The above commands represent a diamond for cursor movement in 8 different directions.

P	Plot point at current cursor
E	Unplot (erase) point at current cursor
=	Draw a line from last point to cursor
-	Erase a line from last point to cursor
F	Fill area from last point to cursor
C	Clear area from last point to cursor
D	Change device path
L	Load File at current address
S	Save File
A	Change address: See below for ranges
T	Transfer character from another set
1	Replot the grid
G	Toggle grid plotting on and off
Z	Zero out (clear) current character
!	Invert current character
?	Display the help screen
R	Read new block of characters to screen
Q	Quit the editor and exit program

*8	Set for 8 by 8 operation
*6	Set for 16 by 16 operation
*B	Change blocksize (max 6x4, min 2x2)
*W	Write as binary file (bsave)

Memory Management: The graphics editor defaults to an address of $4f00 (the normal graphics buffer). Alternate graphics buffers can be used in the text area (starting at $8800). Each graphic file is $fe0 bytes long. The highest you can set the address is $300 bytes below the screen display (which is $BC on 48K and $9C on 40K). For character editing the default is $2C00 which represents the block characters ($400 long). Alternative addresses are the text buffer, with the highest address setting allowed of $300 bytes below the screen display. Lengths of character sets are: $400 for blocks and Normal Atari sets, $300 for 8x8 movie credit fonts and $C00 for 16x16 movie fonts.

Graphics from the graphics editor are always saved as binary files. Fonts from the character editor are saved as data files using the Save option. This is how you should save your font files so they will be properly accessible from the font adjuster and the font builder. The block characters, however, should be saved as binary files with the Write command. In this way they can be set up as bootable files (saved as MC.BLK) or accessed during program operation with the bload command.

Under the execution protion of Movie Credits, all numeric parameters not entered are substituted with zeros and all parenthesis not entered are added. The program makes every attempt to read your mind and operate even if all the required information is not given. If a syntax error occurs (from a non-found verb), execution will stop and bell will sound twice signalling the error (we don't wish to see error messages on our video tapes)! If execution prematurely stops, it means we have run into that situation and should correct the offending word where our execution stopped. Under the edit mode the error messages are:

Syntax - A word was found that did not match our command table. This is usually a typographical error.

Range - A numeric range error occured. This can often be the lower linenumber being larger than the higher number as in list 44,33 or a number exceeding the range of +-32767.

Past End - An attempt was made to edit or move or insert past the end of the file such as Ins 45 when there are only 30 lines in the file.

Wrong Type - An attempt to bload a text file or the like.

Locked - An attempt to write to a locked file or delete it.

Memory Full - A file is too big to fit in memory (under load, so it will not be loaded) or the addition of the current line from the editor under add mode.

Full - The current disk does not have room for the file being written to it or there is no space for directory entries left on the disk.

Bad Path - The device selected does not exist or cannot go to that density.

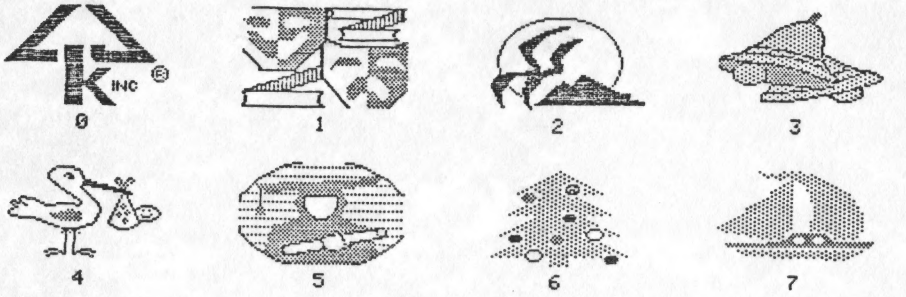
Structure - The file structure has been damaged. The sectors accessed may or may not be to this file. Something has gone wrong with the layout of the current disk and all files may be suspect.

Not Found - The filespec given does not exist on this drive.

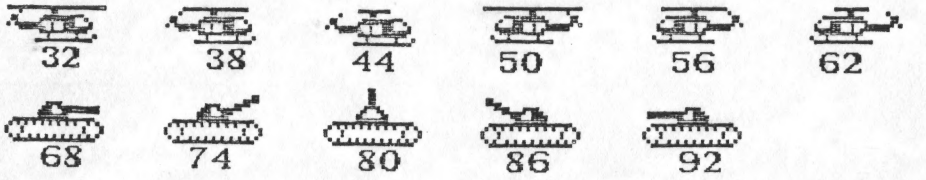
I/O Code: nnn - There was a physical i/o problem. The number given (nnn) is the standard Atari i/o error code. In general, there is something physically wrong with the disk being accessed.

Figure 11

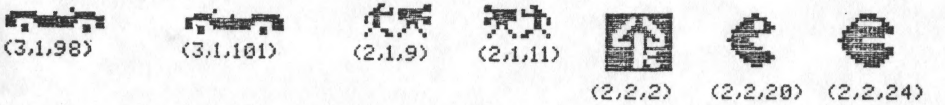
80 x 48 Bit Image Graphics



3 x 2 Character Animation Blocks



Other Animation Blocks



Entire Block

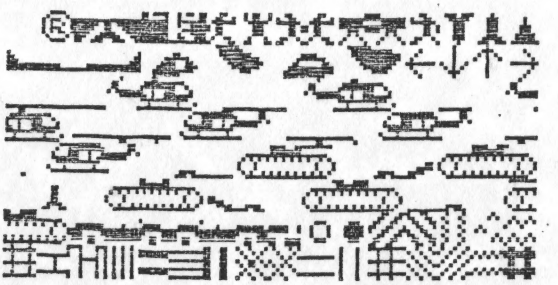


Figure 12

Supplied Fonts

8 - Gothic 8 Aa1©↖

1 - Script 8 Aa1©↖

2 - Old English 8 Aa1©↖

3 - Cyber 8 Aa1©↖

4 - Roman 12 Aa1©↖

5 - Typewriter 12 Aa1©↖

6 - Gothic 16 Aa1©↖

7 - Broadway 16 Aa1©