Texas Instruments
TI-99/4A Home Computer

SUPER Extended BASIC

FOR THE TI-99/4A HOME COMPUTER

CONTENTS: ***SUPER Extended BASIC Cartridge***
TI Extended BASIC Manual
SUPER Extended BASIC Manual

TABLE OF CONTENTS

TABLE OF CONTENTS

INTRODUCTION

Along with the standard TI Extended Basic module, this module contains an added set of enhancements. These enhancements have been designed to aid the Extended Basic programmer.

These include added commands to aid in program editing, modifications to some existing commands to aid in program debugging and added CALLs. Also added is a disk catalog routine to easily obtain a catalog of files on a diskette without leaving Extended Basic.

The following enhancements were made by D.C. Warren for the MG Gram Kracker; Call Clock, Call Clkoff, Call New, Call Bye, Call Clsall, Call Cat and Auto-Load bypass.

The following enhancements were made by Danny Michael for the MG Gram Kracker; Call Init, List, Call Load, Res, Trace, Error Messages, Enhanced Auto-Load bypass, New cursor controls, Copy, Move, Del, Call Peekg, Call Pokeg, Call Peekv, Call Pokev, Call Quiton and Call Quitoff

The following enhancements were made by Mike Dodd for the MG Gram Kracker: Cursor Control and additional deletes, Call All, Call Alock, Call Beep, Call Chimes, Call Colors, Call Ctrl, Call Fctn, Call Gosprt, Call Stsprt, Call Gosub, Call Goto, Call Honk, Call Keys, Call Runprog, Call Scroff, Call Scron, and Call Shift.

The Call Drawnplot routines are from Quality 99 Software's Draw 'N Plot program that was written by M. Shillingburg.

MODIFIED COMMANDS

The following commands, keywords, and subprograms have been modified:

CALL VERSION
This CALL now returns 120 as the version number instead of 110.

CALL INIT
In the old Extended Basic, CALL INIT loaded more data into expansion RAM than necessary. This has been corrected.

LIST
The Extended Basic LIST routine has been modified to allow you to specify a line length when you are outputting your listing to a device such as a printer. The following syntax is used:

LIST *"device name" : line length : start line - end line*
Examples LIST "PIO":28:100-250
LIST "RS232.BA=1200":132:

You can specify a line length ONLY if you also specify an output device. A colon must follow the line length. If not included in the LIST command, the line length is set to the default of the specified output device, usually 80. The line length can range from 1 to 255. If the length specified is outside this range, a Bad Line Number Error is generated. This is due to the routine used to detect the number.

CALL LOAD
This subprogram, when used to 'poke' data into RAM, i.e., CALL LOAD(8192,15), no longer checks to see if you've done a CALL INIT. If used to load an assembly object code file, the CALL INIT check is still made. This now allows you to 'poke' data into scratch pad ram without memory expansion.

PERMANENT
This keyword, which could be used in an OPEN statement has been deleted. This is really no loss as its use had no effect on a file anyway.

RESEQUENCE
This command has been deleted. You must use the abbreviation RES to renumber a program.

RES
Modifications have been made to the RES command which allow you to resequence a portion of the program. Also, please note that the new RES routine DOES NOT replace undefined line numbers with 32767. Any undefined line numbers in the program are left as is. The new command format is:

RES *initial line , increment , start line - end line*

MODIFIED COMMANDS Cont

If you want to use the default values for initial line (100) and increment (10) then you must include the two commas before specifying the starting line. The following commands are all legal:

RES 20,1,10-50	Lines 10 thru 50 are renumbered. Line 10 becomes 20, and the increment is 1.
RES ,5,700-800	Lines 700 thru 800 are renumbered. Line 700 becomes 100, and the increment is 5.
RES ,,50-80	Lines 50 thru 80 are renumbered. Line 50 becomes 100 and the increment is 10.
RES 1000,,750-	Lines 750 thru the last line in the program are renumbered. Line 750 becomes 1000 and the increment is 10
RES ,20,-400	All lines thru 400 are renumbered. The first program line becomes 100 and the increment is 20.
RES 20,,40	Line 40 is renumbered to 20.

RES cannot be used to move lines from one location to another inside the program. If the new line numbers generated by the RES would result in a line being moved, a Bad Line Number Error is generated. A Bad Line Number Error is also given if there are no valid program lines between starting line and ending line.

TRACE
This command has been modified to allow the trace output to be sent to any output device, such as a printer. In order to send the TRACE to a device, you must OPEN file number 123 with the desired output device. This MUST be done in a program line. You must also turn the TRACE mode on, either in command mode or within your program. For example, if the following line is in your program,

100 OPEN #123:"PIO"

and the TRACE is turned on, then the trace output for all lines executed after line 100 will be sent to the PIO port. All lines executed up to and including the line containing the OPEN #123 statement will have their trace numbers printed on the video monitor as in the usual TRACE. It is recommended that you put the OPEN #123 statement in the first line of your program and then use the TRACE and UNTRACE commands to turn the trace on and off. In a nutshell, the TRACE command has been modified to look for an open file #123 before printing the trace number on the screen. If that file IS open, then the trace number is sent to the device specified in the OPEN statement. This may seem to be a backwards way of

implementing this feature (as opposed to TRACE("PIO")), but it does offer a lot of flexibility in the trace printout. Consider the following example:

```
1 OPEN #123:"PIO" :: PRINT #123:"Trace of main program"
100 ! Main
110 ! program
130 ! goes
140 ! here.
150 CALL EXAMPLE
160 ! More
170 ! program
180 ! here
190 END
200 SUB EXAMPLE :: PRINT #123:"Trace from subprogram EXAMPLE"
210 ! subprogram
220 ! goes
230 ! here
240 PRINT #123: :"Returning to main program" :: SUBEND
```

As you can see, having the trace file available to your program can be an advantage.

One disadvantage to sending the trace to a printer is the fact that very few printers will print a line a few characters at a time. For this reason, the trace routine will store the output until a full print line is accumulated. Due to this, several lines of your program will execute before their numbers are printed. You can speed this up a little by specifying a short record length in the OPEN #123 statement.

```
1 OPEN #123:"PIO",VARIABLE 40
```

will cause the trace output to print when 40 characters have been accumulated. (It will also only use half the width of your paper.) NEVER SPECIFY A RECORD LENGTH LESS THAN 9!! The results are unpredictable if you do! While on the subject, the file should always be opened in DISPLAY format, VARIABLE record lengths, and in an output mode (OUTPUT, UPDATE, or APPEND). Since UPDATE, DISPLAY, VARIABLE are the default settings, the only time you should specify anything is if you want a record length other than the default of the device specified in the OPEN statement.

The trace buffer is printed whenever any of the following occurs:

The next traced line number text would exceed the record length.

The program ends.

The UNTRACE command is executed. UNTRACE does not close file #123.

A program error is encountered.

A breakpoint is reached.

You stop the program with the BREAK (FCTN 4) key.

Since the break key will create an error if outputting to the RS232 card, a routine has been added that waits for you to release the break key before printing the trace buffer. This wait routine functions if file #123 is open, whether or not the TRACE is turned on.

If your program closes file #123 any lines executed after the CLOSE statement will be traced to the screen. It is recommended that you first print to file #123 before closing it to ensure that the buffer contents are printed. Use the following line to close the file:

```
PRINT #123: :: CLOSE #123
```

One last feature of the new TRACE routine. If the current line being traced is not the next physical program line after the last traced line, an asterisk is printed prior to the current line number. This only applies if file #123 is open.

NEW FEATURES

CHARACTER PATTERNS AND ERROR MESSAGES
The cursor is redefined to an underline. Error messages are changed from all upper case to upper/lower case.

AUTO LOAD BYPASS
Upon entering Extended Basic, the attempt to run a program named LOAD from DSK1 can now be bypassed. By holding down any alphanumeric key on the keyboard while entering Extended Basic, the load attempt is bypassed. Any key includes the numeric key pressed to select Extended Basic from the main menu screen. If you do not wish to bypass the auto load, you must release the menu selection key quickly.

CURSOR MOVEMENT
Additional cursor control has been added when entering or editing a program line. These new movements are implemented by pressing the SHIFT key along with the FCTN key and an arrow key.

FCTN-SHIFT S	-(left arrow) will return the cursor to the beginning of the input line.
FCTN-SHIFT D	-(right arrow) will position the cursor after the last non-space character in the line.
FCTN-SHIFT E	-(up arrow) will move the cursor up one line, provided the cursor is not already in the first screen line of the input.
FCTN-SHIFT X	-(down arrow) will cause the cursor to move down one line, provided the cursor is not already in the last screen line of the input.
CTRL S	-(left arrow) tabs the cursor 5 characters to the left.
CTRL D	-(right arrow) tabs the cursor 5 characters to the right.
CTRL W	-Word Tab, tabs to the first non-blank character following the next space character(s).
CTRL C	-Deletes all characters from the cursor to the end of the edit or input line(s), including the character underneath the cursor.
CTRL Z	-Deletes all characters from the cursor to the beginning of the edit or input line(s), excluding the character underneath the cursor.

These additional cursor movement controls will also work when responding to an INPUT or ACCEPT statement in a running Extended Basic program.

NEW COMMANDS

The following commands and subprograms have been added to Extended Basic.

COPY
The COPY command is used to copy a program line or block of program lines to any other location in the program. The format is:

COPY *start line - end line , new start line , increment*

The original line or lines remain intact. The block to be copied is defined by start line - end line. If either of these numbers are omitted, the defaults are the first program line and the last program line. However, at least one number and a dash must be entered (you cannot omit both), and there must be at least one valid program line between starting line and ending line. To copy one line you must enter it as both the starting and ending line number. If any of the above conditions are not met, a Bad Line Number error will result.

The new starting line number defines the new line number of the first line in the block to be copied. This number must be entered. There is no default. The increment defines the line number spacing of the copied lines and may be omitted. The default is 10.

There must be sufficient space in the program for the copied segment to fit between new starting line number and the next program line following the location where the block will be moved. If not, a Bad Line Number error message is generated. This problem can be corrected by using a smaller increment, or by using RES to open up space for the segment. A Bad Line Number error also results if the copying process would result in a line number higher than 32,767.

The COPY routine does not change any program references to the copied lines. It is an exact copy of the source lines with new line numbers. A check for sufficient memory space is made before each line is copied. If space is not available the copying process is halted and a Memory Full error results. PLEASE NOTE - the COPY command copies the lines in reverse order. If the copying process is halted due to insufficient memory space, any uncopied lines will be at the beginning of the block.

Before the first line is copied, any open files are closed and all variable values are lost.

DEL
This new command will delete a line or group of lines from your Extended Basic program. The format is:

DEL *start line - end line*

Starting line and ending line define the block of lines to be deleted. If starting line is omitted, line deletion will begin at the first line of the program. In this case, ending line must be preceded

by a dash. If ending line is omitted and starting line is followed by a dash, then program lines from starting line through the end of the program will be deleted. At least one valid program line must exist between starting line and ending line. If not, a Bad Line Number error will result. If only one number is given, without a dash, then that one line will be deleted, if it exists. If it does not exist, a Bad Line Number error is generated.

After the DEL command has executed any open files are closed and all variable values are lost.

MOVE
The MOVE command is used to move a program line or block of program lines to another location in the program. The format is:

MOVE *start line - end line , new start line , increment*

The block of lines to be moved is defined by starting line and ending line. If either of these numbers are omitted, the defaults are the first program line and the last program line. However, at least one number and a dash must be entered (you cannot omit both), and there must be at least one valid program line between starting line and ending line. To move one line you must enter it as both the starting and ending line number. If any of the above conditions are not met, a Bad Line Number error will result.

The new start line number defines the new line number of the first line in the moved segment. When the block is moved it will be renumbered and all references to those lines in the program will be changed to reflect the new line numbers. The new starting line number MUST be entered. There is no default. The increment defines the line number spacing of the moved lines, and may be omitted. The default value is 10.

There must be sufficient space in the program for the moved segment to fit between new starting line number and the next program line following the location where the block will be moved. If not, a Bad Line Number error message is generated. This problem can be corrected by using a smaller increment, or by using RES to open up space for the segment. A Bad Line Number error also results if the renumbering process would result in a line number higher than 32,767.

Although moving lines within the program does not increase the size of the program, this new command requires 4 bytes of program space for each line to be moved. This memory use is temporary, but it must be available in order to move the block. If sufficient memory is not available a Memory Full error results and no lines are moved. This problem can usually be worked around by moving the block a few lines at a time.

Before the block is moved any open files are closed and any variable values are lost.

CALL ALL*(numeric exp)*
This CALL rapidly fills the screen with the character designated by the numeric variable or number in the parenthesis. CALL ALL(32) is the same as CALL CLEAR. CALL ALL(42) fills the screen with asterisks.(*).

CALL ALOCK*(numeric variable)*
Reads the position of the Alpha Lock key and returns a 1 if it is down and a 0 if it is up.

CALL BEEP
Generates a standard beep or accept tone.

CALL BYE
Acts the same as BYE so it can be used in a running program.

CALL CAT(*"device."*)
Generates a file catalog on the screen for a floppy disk drive or a Ram disk. The device MUST have a standard cataloging routine built into its DSR Roms. CALL CAT("DSK1.") will catalog the diskette in drive 1. This CALL can only be executed from command mode. It can not be used in a running program.

CALL CHIMES
Plays a chimes sound, like the one in the Editor Assembler manual.

CALL CLOCK
Displays a 24 hour clock in the upper right hand corner of the screen. This routine is Interrupt Driven, so it will slow down and lose time whenever the disk drive is accessed. NOTE: This CALL requires Memory Expansion.

CALL CLKOFF
Turns off the above (CALL CLOCK) 24 clock.

CALL CLSALL
This CALL closes ALL open files regardless of the file number. This allows you to easily close any and all open files with just one CALL.

CALL COLORS(*f,b*)
This CALL sets the foreground and background colors for all character sets. CALL COLORS(16,5) would set all character sets to white on blue. This CALL does not set the border color, to do this you must use CALL SCREEN(x). If you set the background color in CALL COLORS to 1, transparent, it will allow the CALL SCREEN color to be the background color.

CALL CTRL(*numeric variable*)
Reads the position of the CTRL key and returns a 1 if it is pressed down and a 0 if it is up.

CALL FCTN(*numeric variable*)
Reads the position of the FCTN key and returns a 1 if it is pressed down and a 0 if it is up.

CALL GOSPRT
This is the opposite of CALL STSPRT. CALL STSPRT stops ALL sprite motion and CALL GOSPRT starts them moving again. See CALL STSPRT for additional information.

CALL GOSUB(*numeric variable*)
Executes a GOSUB command, but this allows the use of numeric variables to designated the line number. Example: A = 640 :: CALL GOSUB(A) will execute the subroutine starting at line number 640. Note: Be careful when you use RES since RES will not change the numbers following numeric variables, i.e., A = 640.

CALL GOTO(*numeric variable*)
Executes a GOTO command, but this allows the use of numeric variables to designated the line number. Example: A = 450 :: CALL GOTO(A) will cause your program to jump to line 450 and continue execution. Note: Be careful when you use RES since RES will not change the numbers following numeric variables, i.e., A = 450.

CALL HONK
Generates the standard honk sound.

CALL KEYS(*"keylist",numeric variable*)
This CALL allows you to do a CALL KEY with a validated key list. It will WAIT until a VALID key is pressed and then return the position of the key in the keylist. If the key pressed is not in the keylist a honk sound is produced and the routine continues to scan the keyboard.

Example: CALL KEYS("ESDX",P) - will return the position of the valid key pressed. So, E will return 1, D will return 3 etc. This allows you to easily use the numeric variable in the ON P GOTO xxx,xxx,xxx,xxx or ON P GOSUB xxx,xxx,xxx,xxx commands such as;

```
200 K$="ESDX" :: CALL KEYS(K$,P):: ON P GOSUB 300,400,500,600
300 ! up arrow program segment
400 ! left arrow program segment
500 ! right arrow program segment
600 ! down arrow program segment
```

Note: If you include CALL KEY(3,K,S) in the beginning of your program, this routine will NOT be case sensitive. This will allow both B and/or b to be pressed when only B is in the keylist.

CALL NEW
This acts the same a NEW but allows you to use it in a running program. This provides a method of ending a program and having it cleared out of memory.

CALL PEEKG(*grom address,numeric variable list*)
This subprogram reads data from GROM or GRAM into the variable(s) specified. It functions identical to the regular PEEK subprogram except that it reads from GROM or GRAM.

GROM/GRAM addresses above 32,767 must be converted to a negative number by subtracting 65,536 from the desired address.

CALL POKEG(*gram address,value list*)
This subprogram writes the data in value list to GRAM at the specified address. It functions identical to CALL LOAD except that it writes to GRAM. Assembly language object files cannot be loaded with CALL POKEG. Use CALL LOAD for that. GRAM addresses above 32.767 must be converted as described in CALL PEEKG.

Note: The Super Extended Basic module does NOT contain any GRAM. CALL POKEG was included only for compatibility with programs written for the MG GRAM KRACKER. Its use with this module could cause your program and/or computer to lock up, requiring you to turn off your computer for a moment to regain control.

CALL PEEKV(*vdp address,numeric variable list*)
This subprogram reads data from VDP RAM into the variable(s) specified. It functions identical to the regular PEEK subprogram except that it reads from VDP.

The VDP address should not exceed 16,383.

CALL POKEV(*vdp address,value list*)
This subprogram writes the data defined in value list to VDP RAM at the specified address. It functions identical to CALL LOAD except that it writes to VDP. Assembly language object files cannot be loaded with CALL POKEV. Use CALL LOAD for that.

The VDP address should not exceed 16,383.

CALL QUITOFF
Disables the QUIT (FCTN =) key. This prevents an accidental QUIT from happening when you only wanted and = or +. It also prevents abrupt ending of your programs, without closing the open files.

CALL QUITON
Makes the QUIT (FCTN =) key functional. There are no optional parameters. This CALL has no effect unless you have previously used CALL QUITOFF.

CALL RESTORE(*numeric variable*)
This works the same as RESTORE, except this CALL allows you to use a numeric variable to designate which line number the data pointer will be restored to.

CALL RUNPROG(*"device.filename"*)
This functions the same as RUN except that it now allows you to use string variables in a running program to specify the device and filename. TI Extended Basic does not allow RUN A$, so use CALL RUNPROG(A$) in your programs. This CALL along with CALL KEYS will allow you to easily build a menu of your favorite XB programs and run them with a single key press.

CALL SCROFF
This will turn off the screen like the screen time out does except a key press will not turn it back on (see CALL SCRON). This allows you to turn off the screen, build your screen display and then use CALL SCRON to turn on the screen back on for a rapid display.

CALL SCRON
This is the opposite of CALL SCROFF. It turns the screen back on after you have used CALL SCROFF in a running program.

CALL SHIFT(*numeric variable*)
Reads the position of the SHIFT key(s) and returns a 1 if it is pressed down and a 0 if it is up.

CALL STSPRT
This CALL disables all automatic sprite motion. In other words it prevents ALL sprites from moving. This allows you to easily use multiple sprites to make up on larger graphics character and then put them all into motion at the same time (see CALL GOSPRT). This will allow all of the sprite to move together so they do not break apart on the screen.

CALL GOSPRT
This is the opposite of CALL STSPRT. This will restart or start automatic sprite motion for ALL sprites at the same time.

CALL DRAWNPLOT

This is a special assembly language subprogram. It requires Memory Expansion in order to run. It also requires CALL FILES(2), NEW and CALL INIT to be executed prior to CALL DRAWNPLOT.

When you execute CALL DRAWNPLOT approximately 6K of assembly language subprograms are copied out of the Super Extended Basic module into Low Memory Expansion. These routines fill up virtually all of Low Memory Expansion so they will write over any other assembly program(s) loaded there, except for the items placed there by CALL INIT

GENERAL DESCRIPTION
CALL DRAWNPLOT loads fifteen assembly language subprograms into Low Memory Expansion. They incorporate Hi-resolution graphics capabilities for the Extended Basic programming language.

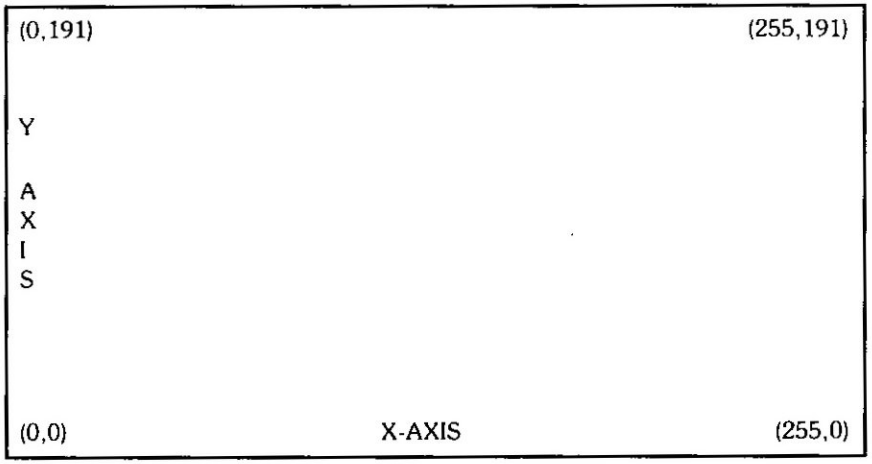
With these routines you may initialize an area of memory (clear it) to be used as a plotting surface, move an imaginary 'Pen' with or without drawing a line, draw boxes and circles and add labels to create your own custom plots, charts and graphs for your programs. Also included are subprograms that permit you to directly edit plots with the #1 joystick and to show your plots on the screen.

Seven of these subprograms enable you to easily dump your plots to a printer and save or load graphics to or from other storage mediums. (disk, RS232, or other devices.)

MINIMUM SYSTEM
This software requires the TI 99/4A console (99/4 will not support these routines) and memory expansion. One or more disk drives and RS232, compatible printer and joysticks are optional. It can only print to an Epson, IBM, or Prowriter Graphics Compatible printer, e.g., TI Impact, Epson, Star, Gemini, Panasonic, Prowriter, etc.

BASIC PRINCIPLES

These subprograms create and maintain a 'plotting surface' in the memory expansion for display on the screen. The plotting area is 256 dots horizontally by 192 vertically. This may be thought of as a coordinate system with the origin (0,0) at the lower left-hand corner of the screen. Thus, the X-axis (horizontal range) is from 0 to 255 and the Y-axis (vertical range) is from 0 to 191. Each dot can be accessed by using it's two coordinate values (x,y).



These routines utilize the bit-map mode of the TMS 9918A video processor. Because of the large amount of memory that is required by the 9918A in this mode, and the operation of Extended Basic, a few compromises had to be made. Your plot or graph can only be viewed with the 'show' or edit subprograms and no more than two files can be open at any one time throughout your programs (CALL FILES(2)). The following pages describe each of the subprograms, their syntax, parameters and results.

These routines may be used in command mode or in your program.

Before using any of the subprograms on the following pages you must first execute the following items:

```
CALL FILES(2)
NEW
CALL INIT
CALL DRAWNPLOT
CALL LINK("GCLEAR")
```


CALL LINK("GCLEAR")

This subprogram clears the graphics area and moves the 'Pen' to the origin (0,0) at the lower left-hand corner. This routine also loads the ASCII character set into memory for use in the label subprogram. Thus, THIS SHOULD ALWAYS BE USED BEFORE ANY OF THE OTHER PLOTTING SUBPROGRAMS ARE LINKED TO.

CALL LINK("MOVE",*X-position,Y-position*)

This subprogram moves the imaginary 'Pen' from its present location to the location specified WITHOUT drawing a line. The location is given in the parameters that are passed with the subprogram CALL. The X and Y positions are numeric expressions or variables. NOTE: If the point specified in the move or draw routines is off the screen, the 'Pen' moves off the screen but a draw will only draw a line to the edge of the screen. Although there is no risk of destroying data by moving or drawing off the screen, there are dangers of numeric overflow.

CALL LINK("DRAW",*X-position,Y-position*)

This subprogram will move the 'Pen' and DRAW a line from it's present location to a point specified in the parameters passed. The restrictions are the same as for the move command. (See note above)

CALL LINK("CIRCLE",*X-position,Y-position,R-radius*)

This subprogram draws a circle in your drawing or plot. The center of the circle will be located at the point (X,Y), where X and Y are numeric expressions. The circle will have a radius of R. The radius must be a numeric expression not equal to zero and have an absolute value less than 128. If the radius is zero or more than 127, the pen will be moved to the center coordinates. Otherwise, a circle will be drawn and the pen will be moved to the center coordinates. NOTE: A circle that is centered off the screen may have unpredictable results.

CALL LINK("SQUARE",*X-position,Y-position,S-length*)

This subprogram draws a box in your drawing or plot. The variables X and Y are the lower left hand corner of the square, and S is the length of a side. After the square is drawn, the pen will be returned to the position (X,Y). The maximum length side is 256 pixels (dots).

CALL LINK("LABEL",*string-exp*)

This subprogram allows the user to place labels made up of ASCII characters from char 32 to char 127. These may be the standard ASCII characters or others that have been defined with the CALL CHAR subprogram prior to the use of the GCLEAR subprogram. The string is placed starting at the character block in which the 'Pen' is currently located (X,Y).

CALL LINK("SHOW")

This subprogram is used to display the plot or drawing on the screen. It initializes bit map mode and then moves your plot out to VDP memory from its buffer in High Memory Expansion. To exit show, simply press 'E'.

The use of SHOW and EDIT subprograms have some limitations and restrictions that should always be remembered:

1. These commands must not be located within the bounds of any Extended Basic subroutines or subprograms. If this happens an error may be issued.
2. These subprograms over write 12,000 bytes of VDP memory when bit map mode is initialized. Therefore, there may be a loss of string data.
3. Due to this extensive use of VDP memory there is room for only two I/O files to be open at any one time throughout program execution, CALL FILES(2).

CALL LINK("EDIT")

This subprogram allows you to edit the plotting area directly with the #1 joystick and the keyboard. When this subprogram is called a flashing dot will appear at the last X,Y position or the 0,0 position after a GCLEAR. This is the 'Pen' point. This flashing dot, which is a cursor of sorts, can be moved in any one of eight directions. If you desire to draw a line while moving, simply press the fire button on the #1 joystick.

The keyboard also plays an important role while editing because this is a full function graphics editor capable of not only drawing but also erasing. On the following page is a listing of the keys used in the editor.

EDIT KEY LIST

C - Clears the plotting surface.

L - Displays the Label prompt, a question mark (?), at the current position and permits you to type ASCII characters onto the screen. In this mode both upper and lower case are active along with the right and left arrow keys (FCTN-S,FCTN-D). When your label is complete simply press enter to return to the 'Pen' mode.

D - Draws a line from a previous point to the present cursor position, this allows you to draw connecting lines with ease. A new previous point can be marked by using the M key or the firebutton on the #1 joystick. (see Mark below.)

F - This key makes the pen move faster.

S - This key returns the pen to its normal slower speed.

W - This is the write/erase toggle. When the user enters the editor it is always in the write mode. To enter the erase mode press W and use the #1 joystick and fire button to erase. To return to the write mode press W again. Thus, it will change modes each time W is pressed.

M - This key marks the present position of the cursor and stores it as the first point of a line, the lower left hand corner of a box or the center of a circle, that may be drawn using the D, O or B keys. You may also mark a point by pressing the firebutton on joystick #1.

O - Draws a circle. First move the cursor to the desired center of the circle and press M, or the #1 joystick fire button, to mark the center. Next, move the cursor horizontally to the desired radius and then press O (not zero). The circle will be drawn very quickly and the cursor will return to the center of the circle. If the radius is larger than 127 pixels (dots), the circle will not be drawn, but the cursor will be returned to the center position.

B - Draws a square. First move the cursor to the position desired for the lower left hand corner of the square and press M, or the fire button, to mark the spot. Next, move the cursor horizontally to the right for the length of a side and then press B (for box). A perfect square will be drawn instantly and the cursor will be returned to the lower left hand corner.

I - This routine will fill or paint a bounded area on the screen. The fill routine can fill most convex objects (i.e. square, circle, etc.) in one keystroke. In some instances, the shape will not be completely filled. One example of this is the long slender triangle.

To invoke the Fill routine from the editor mode, simply position the cursor (flashing dot) inside of the object, then press "I". If the object is not completely filled, move the cursor to the unfilled portion of the object and press the "I" key again. For Concave objects (odd shaped) repeat the above process until the object is full.

To optimize the fill routine, always invoke it with the cursor in the center of the area to be filled. This will insure the best coverage of the area.

The boundary of the object must be solid. The edges of the screen are considered to be boundaries. If the boundary is not closed, and the Fill routine is invoked, the result of the fill may be unpredictable.

NOTE: The larger the area to be filled, the longer it will take for the fill routine to return control to the editor. For example, to fill the entire blank screen (49,152 pixels) will take approximately 30 seconds.

E - This key is used to exit the editor and return to the main program or Extended Basic command mode.

FIRE BUTTON - Use the fire button to draw or erase lines (with the W key) on the screen. This is achieved by moving the #1 joystick controller and pressing the fire button simultaneously. Make sure that the alpha lock is off so that the joystick unit operates correctly.

Hint: To erase large areas quickly, use the label command, and enter spaces.

Because of the similarity to the show command, the same restrictions apply to the edit command. Please read the notes contained in the description of the show command, they are very important!

PRINTING YOUR DRAWINGS AND PLOTS

These subprograms send your drawing or plot to a printer via your RS232 or PIO ports. They all utilize the dot addressable graphics mode of your printer. Two of the subprograms were designed to work with the TI Impact, Epson or Gemini printers and two of them are for the Prowriter printer.

The output device, in the syntax below, must be a string expression and include all the software switches necessary for that particular device. For example "PIO" or A$ where A$ is equal to "RS232.PA=0.DA=8.BA=9600".

In order to print graphics, RS232 printers must have a DIP switch set to receive 8 data bits. You can also make your printer much faster by setting the baud rate DIP switches to be 4800 or 9600. Then your printer name becomes: RS232.BA=4800.DA=8 or RS232.BA=9600.DA=8. (Note: some printers may require 2 stop bits at these higher baud rates and/or no parity). The screen has 256 dots across, and a printer only has 480 dots across, therefore a bigger picture (full page) would not fit the paper horizontally (256x2=512) so these full page plots are rotated. Most printers do not have equal spacing between dots horizontally and vertically. That makes circles on the screen look like ovals on the paper. Gemini 10X-all DIP switches off, except #2.

CALL LINK("GDUMP","*device*")
TI Impact, Epson or Gemini compatible - Normal Size
device = "PIO" or "RS232", etc.

CALL LINK("LDUMP","*device*")
TI Impact, Epson or Gemini compatible - Full Page

CALL LINK("PWDUMP","*device*")
Prowriter compatible - Normal Size

CALL LINK("PLDUMP","*device*")
Prowriter compatible - Full Page

SAVING YOUR DRAWINGS AND PLOTS

CALL LINK("GSAVE","*device.filename*")
This utility permits the user to save his plot to the device and file name specified. On diskettes the file will use exactly 25 sectors.

The device may also be an RS232 or PIO port. This utility may be used to transfer the direct plotting data to another device via the RS232 port. Note: CS1 or CS2 are not allowed.

CALL LINK("GLOAD","*device.filename*")
This utility allows the user to load previously saved plots (saved with the GSAVE utility). Note: CS1 or CS2 are not allowed.

ERROR MESSAGES:

If an I/O error occurs during GLOAD, GSAVE, or GDUMP, then the message "***I/O error***" will appear at the bottom of the screen. This error message will be issued for the following types of file errors: BAD DEVICE NAME, DEVICE OR FILE WRITE PROTECTED, INCORRECT FILE ATTRIBUTES, INCORRECT OPEN MODE, DISK FULL, DEVICE ERROR, FILE ERROR.

When an error occurs, the image in memory is not altered. When the reason for the error is corrected, the operation can be repeated.

NOTES:

- Do not use the SHOW, EDIT, GDUMP, GSAVE, or GLOAD routines in subroutines or subprograms.
- The plot area takes 6,144 bytes of memory. Therefore, your program must not be larger than 17,000 bytes or the plot will overwrite your program.
- To change the screen and line colors for EDIT and SHOW use these CALL LOADS after CALL DRAWNPLOT has been executed:
  CALL LOAD(9481,screen_color-1)
  CALL LOAD(11420,(pen_color-1)*16)
- For programming examples examine the Menu and Border programs on the following pages.

DRAW 'N PLOT MENU PROGRAM

```
100 !DRAW PROGRAM, QUALITY 9
9 SOFTWARE
110 CALL PEEK(-31888,A,B)::
IF A=57 AND B=221 THEN 130 E
LSE CALL CLEAR
120 PRINT "You must execute"
:"CALL FILES(2)    NEW":"befo
re using CALL DRAWNPLOT": :
: : : : :: END
130 CALL INIT :: CALL DRAWNP
LOT :: CALL LINK("GCLEAR")
140 CALL CLEAR :: CALL SCREE
N(3):: CALL COLORS(3,16):: D
ISPLAY AT(3,7):"Draw 'N Plot
(tm)"
150 DISPLAY AT(6,1):"Press
      To": :"  1      Create a
new image": :"  2      Load im
age from disk": :"  3      Sho
w image in memory"
160 DISPLAY AT(14,1):"  4
Save image to disk": :"  5
Print image": :"  6      Ch
ange colors": :"  7      Exit"
170 DISPLAY AT(23,6):"QUALIT
Y 99 SOFTWARE"
180 CALL KEYS("1234567",K)::
ON K GOTO 260,190,330,350,4
10,290,580
190 DISPLAY AT(3,7)ERASE ALL
:"Draw 'N Plot (tm)"
200 DISPLAY AT(23,6):"QUALIT
Y 99 SOFTWARE"
210 DISPLAY AT(10,1):"ENTER
FILENAME": :"    (I.E. DSK1.C
ASTLE)": : : :
220 ACCEPT AT(14,1):AA$
230 PN$="A"
240 CALL LINK("GLOAD",AA$)!L
OAD IMAGE FROM DISK
250 GOTO 270
260 CALL LINK("GCLEAR")
270 CALL LINK("MOVE",107,107
):: CALL LINK("EDIT")!DRAW W
ITH JOYSTICK #1
280 GOTO 140
290 DISPLAY AT(3,7)ERASE ALL
:"Draw 'N Plot (tm)"
300 DISPLAY AT(23,6):"QUALIT
Y 99 SOFTWARE"
310 DISPLAY AT(10,1):"ENTER
SCREEN COLOR, 1-16": : : : :
: ACCEPT AT(13,1)VALIDATE(DI
GIT):AA :: CALL LOAD(9481,AA
-1)
320 DISPLAY AT(10,1):"ENTER
LINE COLOR, 1-16": : : : ::
ACCEPT AT(13,1)VALIDATE(DIGI
T):AA :: CALL LOAD(11420,(AA
-1)*16)
330 CALL LINK("MOVE",107,107
):: CALL LINK("EDIT")
340 GOTO 140
350 DISPLAY AT(3,7)ERASE ALL
:"Draw 'N Plot (tm)"
360 DISPLAY AT(23,6):"QUALIT
Y 99 SOFTWARE"
370 DISPLAY AT(10,1):"ENTER
FILENAME": :"    (I.E. DSK1.P
ICTURE2)"
380 ACCEPT AT(14,1):AA$
390 CALL LINK("GSAVE",AA$)!S
AVE IMAGE TO DISK
400 GOTO 140
410 DISPLAY AT(3,7)ERASE ALL
:"Draw 'N Plot (tm)"
420 DISPLAY AT(23,6):"QUALIT
Y 99 SOFTWARE"
430 DISPLAY AT(10,1):"ENTER
PRINTER NAME": :"    (I.E. PI
O)": : : :
440 ACCEPT AT(14,1):PN$ :: I
F LEN(PN$)  3 THEN 140
450 IF KK    0 THEN 480
460 DISPLAY AT(10,1):"PRESS
1 FOR TI,EPSON,GEMINI": :"PR
ESS 2 FOR PROWRITER": : : :
470 CALL KEYS("12",KK)
480 DISPLAY AT(10,1):"PRESS
1 FOR NORMAL SIZE": :"PRESS
2 FOR FULL PAGE": : : :
490 CALL KEYS("12",YY)
500 DISPLAY AT(10,1): :"
PRINTING": : : : : :
510 IF YY=2 THEN 550
520 IF KK=2 THEN 540
```

DRAW 'N PLOT MENU PROGRAM Cont.

```
530 CALL LINK("GDUMP",PN$)::
GOTO 140 !EPSON NORMAL
540 CALL LINK("PWDUMP",PN$):
: GOTO 140 !PROWRITER NORMAL
550 IF KK=2 THEN 570
560 CALL LINK("LDUMP",PN$)::
 GOTO 140 !EPSON FULL PAGE
570 CALL LINK("PLDUMP",PN$):
: GOTO 140 !PROWRITER FULL P
AGE
580 DISPLAY AT(10,1)ERASE AL
L:"Goodbye:": : :"      Thank
you for using": : :"       Dra
w 'N Plot (tm)": : : :"     Q
UALITY 99 SOFTWARE"
```

DRAW 'N PLOT BORDER EXAMPLE PROGRAM

```
100 !SAMPLE PROGRAM, TO PRIN
T THE MAXIMUM BORDER
110 !
120 !FIRST YOU MUST LOAD DRA
W 'N PLOT (TM) i.e. CALL FIL
ES(2) NEW CALL INIT CALL DRA
WNPLOT
130 !IF YOU DON'T SEE THE EN
TIRE BORDER, THEN YOUR TV IS
OVERSCANNING THE PICTURE. T
RY TO ADJUST THE HORIZONTAL
OR VERTICAL SCAN.
140 !EVEN IF YOU DON'T SEE T
HE ENTIRE PICTURE ON YOUR SC
REEN, THE ENTIRE PICTURE WIL
L BE IN MEMORY, AND
150 !WILL BE PRINTED OUT OR
SAVED TO DISK.
160 !
170 CALL LINK("GCLEAR")!   O
NLY DONE ONCE, AT BEGINNING
OF PGM. PUTS PEN AT 0,0
180 CALL LINK("DRAW",255,0)!
DRAW LINE TO LOWE
R RIGHT CORNER
190 CALL LINK("DRAW",255,191
)!          DRAW LINE TO UPPE
R RIGHT CORNER
200 CALL LINK("DRAW",0,191)!
              DRAW LINE TO UPPE
R LEFT CORNER
210 CALL LINK("DRAW",0,0)!
DRAW LINE TO LOWE
R LEFT CORNER
220 CALL LINK("MOVE",50,100)
!  MOVE PEN TO MIDDLE OF SCR
EEN WITHOUT DRAWING LINE
230 CALL LINK("LABEL","Press
E to print.")!      ENTER TE
XT ON SCREEN
240 CALL LINK("CIRCLE",25,15
0,20)!      DRAW A CIRCLE
250 CALL LINK("SQUARE",220,2
0,20)!      DRAW A BOX
260 CALL LINK("SHOW")!
DISPLAY SCREEN
270 CALL LINK("LDUMP","PIO")
!           FULL PAGE FOR EPS
ON
280 !CALL LINK("PLDUMP","PIO
")!        FULL PAGE FOR PRO
WRITER
290 !
300 !GOOD LUCK, WE HOPE THAT
YOU ENJOY USING DRAW 'N PLO
T (TM)!
310 !      QUALITY 99 SOFTWAR
E
```

You may obtain the above two programs on disk or cassette (specify which), by sending $3 to: Quality 99 Software, 1884 Columbia Rd. #1021, Washington DC 20009.