THE MISSING LINK

The Ultimate Extended Basic Upgrade

Texaments
53 Center Street
Patchogue, New York 11772

Printed in U.S.A.

IMPORTANT NOTICE

ACKNOWLEDGEMENTS

I would like to express my thanks to the following people, all of whom made significant contributions to The Missing Link.

Harry Brashear, John Wilforth, and Barry Traver all offered good suggestions for improving The Missing Link. Ollie Hebert contributed vast amounts of time and energy. He tested the program, edited the manual, and wrote numerous demonstration programs. My wife Donna was incredibly understanding about the many hours that were put in writing, debugging, and documenting the program.

If you like The Missing Link, it is in part due to the extra polishing made possible by these people.

Harry Wilhelm
Author of The Missing Link

TABLE OF CONTENTS

1. INTRODUCTION

The Missing Link is the ultimate extension for Extended Basic. It enhances Extended Basic to the same degree that Extended Basic enhances console basic. The Missing Link consists of a collection of assembly language subroutines that give the Extended Basic programmer complete access to the bit-mapped graphics mode built into the TI-99/4A computer. These subroutines replace the usual methods of accessing the screen.

No knowledge of assembly language is required to fully utilize THE MISSING LINK. Programs can be written completely in Extended Basic. This means that they are both easy to write and easy to understand. The average TI user can now take advantage of the advanced graphics features that were built into the computer, but never before available without delving into the complexities of assembly language.

The Missing Link includes text operations that can input information or display it on the screen. There is automatic word wrap when displaying text. Text can even be displayed vertically. The character size can be changed, permitting up to 32 rows by 60 columns on the screen. Different sized text can be displayed simultaneously on the same screen.

Cartesian graphics are available for plotting points, lines, circles, and boxes. Turtle graphics can be used with none of the ink and color restrictions found in Logo. Sprite graphics can place up to 32 moving sprites on the screen. Best of all, there are no limits when combining graphics and text on the screen.

A graphics screen dump is always available, and pictures can be loaded or saved in the standard TI-Artist format.

In short, The Missing Link can unleash the power of your computer and make Extended Basic programming fun again.

1.1. Equipment Required

The Missing Link requires the TI-99/4A console, the Extended Basic cartridge, the 32K memory expansion, and a disk drive system. An Epson compatible printer is needed to use the screen dump feature.

1.2. Differences From Extended Basic

The Missing Link will set the screen display to the standard graphics mode whenever an Extended Basic program is not running. It will change to a bit-mapped screen whenever a program is running. This happens automatically, and requires no intervention by the user.

The bit-mapped screen is made up of pixels, which are the smallest dots that can be placed on the screen. There are 192 pixel rows and 240 pixel columns

when using The Missing Link. Screen location 1,1 is at the upper left hand corner, and location 192,240 is at the lower right hand corner. (The screen is less than 256 pixels wide because an 8 pixel wide strip on each side of the screen is unavailable for use.)

Two different color configurations that can be used by The Missing Link:

The 16 color mode gives you access to all 16 colors that can be produced by the computer. The color data for the screen is in the form of 1 pixel high by 8 pixel wide strips. Each strip can have a foreground color and a background color. The eight pixels contained in each strip can be individually "turned on" or "turned off". In other words, they can be set to the foreground color or the background color for that strip. Different strips can have different foreground and background colors, but each strip can contain only two colors. Having the color data in the form of 1x8 strips is somewhat limiting; nevertheless, spectacular full color displays can be produced. This limitation is built into the TMS 9918A video chip. *Unexpected colors in the display are almost always the result of this limitation.*

The 2 color mode provides one foreground and one background color for the entire screen. The 2 color mode is the easiest to work with because there is more stack space and no problems with the 8 pixel long strips of colors.

The Bit-Mapped mode requires that the video memory be configured in a drastically different way than is normally used by Extended Basic. This has several important consequences.

None of the usual methods of putting characters or graphics on the screen will work properly. PRINT, DISPLAY AT, INPUT, ACCEPT AT, HCHAR, VCHAR, GCHAR, SPRITE, and others can be used without an error resulting. However, nothing will appear except some garbled color patches in the upper third of the screen. The Missing Link provides assembly language subroutines to accomplish these same operations using bit- mapped graphics.

Except for those concerned with screen and sprite access, all other Extended Basic program statements and subprograms will function normally.

The bit-mapped screen consumes a great deal of video memory. This memory has to come from somewhere. In this case it is obtained at the expense of the stack space. Fortunately, this is not as drastic as it first appears, because there are still the usual 24488 bytes of memory available for your program.

The stack space is primarily used to contain string data and subprogram names. You may have to adjust your programming style in order to conserve the limited stack space, especially when operating in the 16 color mode. Refer to page 23 of this manual for more information on how to conserve stack space.

In the 16 color mode, the use of named subprograms with names more than eight characters long will cause spurious blocks of color to appear on the screen.

In the 16 color mode, INPUTing from a DISPLAY format disk file will cause spurious blocks of color to appear on the screen. You can avoid this by using LINPUT instead. (The use of INTERNAL format files will present no problem.)

Using TRACE to help with debugging will cause spurious blocks of color to appear on the screen. However, the line numbers become visible when you "break" the program with <Fctn 4>.

After you "break" a running program by pressing <Fctn 4>, you can type CON to continue. The screen will reappear just as it was when the program was interrupted, but the colors will be set to black on cyan, and any sprites will be shown in the unmagnified size.

The "quit" key has been disabled. You must type "BYE" then press <Enter> to return to the master title screen.

The Missing Link is very sensitive to unclean contacts on the Extended Basic module. Lockups are invariably caused by dirty contacts, and cleaning them alleviates the problem.

1.3. Loading The Missing Link

Select Extended Basic from the TI master title screen. Place the program disk into a disk drive (drive number "n"), type RUN "DSKn.TML" and then press <Enter>. You are then asked if you are using a Myarc disk controller. Press "Y" if you do, or "N" if you have either a CorComp or TI disk controller.

The loader for The Missing Link will then give you the option of selecting one, two, or three disk files that can be opened simultaneously. When using a Myarc disk controller you may only select one or two files to be opened, CorComp and TI controller users may select up to three files to be opened. Select a number and then press <Enter>. This performs an equivalent of the CALL FILES(n) operation.

Stack space is directly related to the number of disk files that are opened. Each disk file costs 518 bytes of stack space. Therefore, it is important to choose the smallest number of disk files that will still permit your particular application to run. Also, in order to load or save pictures that are in TI-Artist format, it is necessary that one additional disk file be available besides those explicitly used by your program.

You are then given a choice of using either the "16 color" mode or the "2 color" mode. If full color is not necessary, 3752 bytes of stack space are gained by using the "2 color" mode.

After your selections are made, The Missing Link is configured as instructed, and the computer is returned to you in the immediate mode of Extended Basic. (Once in a great while, an error message in line 230 is issued. This appears to be a quirk of Extended Basic and can be ignored.) The screen color will be changed to light green and the cursor will be transformed into a hollow cursor. This tells you that The Missing Link is loaded and active.

Once The Missing Link is activated, the only way to change the number of disk files or the color mode is to type "BYE" to leave Extended Basic and return to the master title screen. You can then reload the program and choose a different configuration.

Following is a list of the stack space available with the various configurations:

- 424 bytes; 16 color mode with 3 disk files available.
- 942 bytes; 16 color mode with 2 disk files available.
- 1460 bytes; 16 color mode with 1 disk file available.
- 4176 bytes; 2 color mode with 3 disk files available.
- 4694 bytes; 2 color mode with 2 disk files available.
- 5212 bytes; 2 color mode with 1 disk file available.

2. THE SUBROUTINES

The Missing Link contains 32 assembly language subroutines, which can be grouped in the following categories:

- Full Screen Operations
- Pen Commands
- Windows
- Text
- Cartesian Graphics
- Turtle Graphics
- Sprite Graphics
- Perpiheral Access

All of these subroutines are intended to be called from within a running Extended Basic program. No error message results when the subroutines are called from the immediate mode, but no action will occur on the screen.

The subroutines are described in the next sections. The first line of each description shows the correct syntax to use when calling the subroutine. Most of the subroutines require that additional information be included after the name of the subroutine. This information is supplied in the form of a parameter list. Be careful to include these parameters in the order described, and not to mix strings and numbers. Sometimes there are optional parameters. These optional parameters are shown enclosed in brackets. The purpose of each of the parameters in the list is fully described.

Unless explicitly stated otherwise, numbers and strings can be constants, variables, or elements of an array. Numeric values do not have to be integers because The Missing Link will automatically round them up or down.

2.1. Full Screen Operations

```
CALL LINK("CLEAR")
```

This subroutine clears the entire screen by setting all pixels to the background color.

```
CALL LINK("COLOR",foreground_color,background_color)
```

This subroutine changes all the pixels on the screen to the specified color combination. Also, the PENHUE (described in the next section) is changed to the same color combination. The two color codes must be numbers from 1 to 16. Refer to page 99 in the Extended Basic manual for a list of the colors. This subroutine has no effect on the screen color seen at the four edges of the display.

```
CALL SCREEN(color_code)
```

The screen color can be changed by using Extended Basic's CALL SCREEN subprogram. Additionally, any pixel in the display that is set to transparent (color code 1) will appear in the screen color. Refer to page 165 in the Extended Basic manual for more information.

2.2. Pen Commands

The pen commands are used to control the status of the pen. By controlling the pen status, the programmer determines exactly what occurs when the pen touches any pixel while performing all subsequent graphics and text operations. (Text operations always assume that the pen is down, regardless of the actual pen condition.)

```
CALL LINK("PENHUE",foreground_color,background_color)
```

This routine changes the color of the pen to the specified color combination. The color codes must be numeric values from 1 to 16. The penhue then determines the foreground and background colors of the 8 pixel wide strip containing any pixel that is subsequently touched by the pen.

A special case occurs when the penhue is set to transparent on transparent with CALL LINK("PENHUE",1,1). This color combination causes THE MISSING LINK to not change the colors of any pixel touched by the pen. This is useful in circumstances where a previous operation has already set the colors of the pixels. Penhue is only functional in the 16 color mode.

```
CALL LINK("PD")
```

This routine sets the status of the pen to pendown. Any pixel subsequently contacted by the pen will be "turned on" - it will be set to the foreground color. The penhue determines the foreground and background colors of the 8 pixel wide strip containing any pixel that is touched by the pen.

```
CALL LINK("PR")
```

This routine sets the status of the pen to penreverse. Any pixel subsequently contacted by the pen will be inverted. Pixels that are "on" will be turned "off", and pixels that are "off" will be turned "on". The penhue determines the foreground and background colors of the 8 pixel wide strip containing any pixel that is touched by the pen.

```
CALL LINK("PE")
```

This routine sets the status of the pen to penerase. Any pixel subsequently contacted by the pen will be "turned off" -- it will be set to the background color. The penhue determines the foreground and background colors of the 8 pixel wide strip containing any pixel that is touched by the pen.


```
CALL LINK("PU")
```

This routine sets the status of the pen to penup. The pen will "pass over" pixels without changing them. However, the foreground and background colors of each 8 pixel wide strip will still be determined by the penhue as if the pen had actually touched the pixel.

2.3. Windows

A "window" is a rectangular area on the screen that specifies the boundaries where text and graphics are to be displayed. Text will always be displayed within a window. Graphics may be displayed either inside or outside of a window. Only one window can be active at any one time, but several windows may be used on a single screen.

```
CALL LINK("WINDOW"[,row1,column1,row2,column2,1])
```

This routine is used to specify the size of a window. To specify that the window is to be the entire screen, simply omit all the optional values.

Row1 and column1 specify the location of the upper left hand corner of the window. Row2 and column2 specify the location of the lower right hand corner. The rows must be numeric values from 0 to 193 and the columns must be numeric values from 0 to 241. To place a frame around the window, include the optional trailing "1". To draw the frame correctly, the pen should already be set to pendown.

```
CALL LINK("REVWIN")
```

Normally, graphics can only be displayed within the boundaries of the window. Calling this routine reverses the window so that graphics will only be displayed outside the boundaries of the window. Calling this routine a second time restores the normal operation of the window. This routine has no effect when text is being displayed.

```
CALL LINK("FILL"[,row1,column1,row2,column2])
```

FILL is a very versatile routine. It causes the pen to touch each pixel within the specified recangular area. By modifying the pencondition and the penhue, FILL can be used to erase selected areas, set text to inverse video, change colors, and so on.

If the optional values are omitted, this routine will fill the entire window area.

The optional row and column values can be used to specify the size of a smaller rectangle within the window. The row values must be from 1 to 192 and the column values must be from 1 to 240. These values are relative to the upper left hand corner of the window. For example, CALL LINK ("FILL", 2,3,10,11) will fill a rectangle inside the window starting at row 2, column 3 and ending at row 10, column 11. Row1 and column1 must fall within the

window area or an error message will be issued. Row2 or column2 can fall outside of the window. In this case, the fill operation will only proceed as far as the window boundaries. No error message will be issued.

2.4. Text

A program can input numbers and strings up to 255 characters long. Also, numbers and strings can be displayed on the screen. The Missing Link does not restrict the display to 24 rows by 28 columns. Instead, there is pixel by pixel accuracy in placing characters, and there is complete control over the size of the characters. There is even a font size that lets text be printed with 32 rows by 60 columns. By using characters of this size you can place the same amount of text on the screen as is contained in a 24 by 80 column display. When using very small characters, the color combination of dark green on light green (13,4) gives the most legibility.

Only characters having ASCII codes from 32 to 127 can be printed or input. When operating in the 16 color mode, the current penhue determines the color of the characters. The pen condition is always ignored. When characters are displayed, the character is shown in the foreground color atop a rectangle in the background color.

Each character is placed on the screen by erasing a small rectangular block of pixels and then printing the character inside that block. The programmer can modify the width and height of this block of pixels. This is the character size and it directly determines the number of rows and columns that will fit on the screen. Different size characters can be displayed on the screen at the same time.

Characters are displayed sequentially from left to right and from top to bottom. When The Missing Link no longer has room for an entire character on the current line, it drops down a row, goes to the left hand boundary of the window, and continues.

Text and numbers are always printed or input inside a window, which gives the programmer total control over the screen appearance. By using a tall, narrow window it is possible to print text vertically.

When numbers are being displayed there is precise control over the numeric format used.

```
CALL LINK("PRINT",row,column,string_or_number
        [,string_variable])
```

Used to print a string or a number to the screen. A string can be up to 255 characters long. The row and the column are the pixel row and the pixel column relative to the upper left hand corner of the window. Numeric values are printed as specified by the FORMAT routine described below.

Word wrap is always on when using PRINT. The rules are simple:

- Leading spaces are deleted so that the first character on a line will not be a space
- If a word will not fit in the remaining space on the line, then The Missing Link will fill the rest of the line with spaces, drop down a line, go to column 1 of the window, and continue printing. A word will only be broken if it is too long to fit totally within the left and right columns of the window.
- Trailing spaces at the end of a string are not deleted unless The Missing Link has to start a new line.

Sometimes a string or a number has too many characters to completely fit inside the window. In that case, it will be truncated upon reaching the lower right corner of the window. Including the optional string variable will retrieve the portion of the string that would not fit within the window. Your program can then deal with the string fragment as desired.

When printing a succession of strings to the screen, it sometimes is helpful to have a pointer to where The Missing Link left off printing. A text adventure is an example of an application where this would be useful. If the row and column are numeric variables, they will automatically be updated so that they point to the next available character position in the window. If you don't want to have these variables updated, simply enclose either of them in parentheses. No update can occur if the row or column are numeric constants.

```
CALL LINK("INPUT",row,column,string_or_numeric_varable
        [,length,prompt_string])
```

Used to input a number or a string up to 255 characters long from the screen. The row and the column are the pixel row and column relative to the upper left hand corner of the window. The routine assumes that 255 characters are to be input unless you specify an optional length ranging from 1 to 255 characters.

An optional prompt can be specified that will appear on the screen as a suggested response. The prompt must be a string, but it can be used when inputting either a string or a number. If the response is appropriate, the user can simply press <Enter>. Otherwise, the response can either be erased or modified as desired.

The routine first clears a space on the screen long enough to allow the specified number of characters to be input. If the window is too small to permit all the characters to be input, then the window boundaries determine the maximum number of characters. Then the optional prompt, if any, will be displayed, and finally the cursor will appear flashing atop the first character or space.

The keyboard functions are virtually identical to those used by Extended Basic. <Fctn S> and <Fctn D> move the cursor left and right. <Fctn 1> deletes a character. <Fctn 2> is used to insert characters. <Fctn 3> erases from the cursor position to the end of the line. The keys will repeat if held down. Press Enter to input the string or number.

There is no equivalent to Extended Basic's VALIDATE option. However, when inputting a numeric value, only the ten number keys and the "E + - ." keys are active. If no numbers are typed before pressing <Enter>, then the numeric variable will equal zero.

```
CALL LINK("CHAR",ASCII_code,hexadecimal_string)
```

Used to redefine character patterns. With a few differences, this is the equivalent of the CALL CHAR subprogram in Extended Basic.

Only ASCII characters from 33 to 127 can be redefined. The space and the cursor cannot be redefined. The hexadecimal string that defines the character pattern can be up to 240 characters long. This means that up to 15 consecutive ASCII characters can be redefined each time this subroutine is called. When defining a sequence of character patterns, trying to define ASCII characters higher than 127 by using a long hexadecimal string will result in the excess string being ignored. No error message will be issued. If necessary, the computer will add zeros to the string so that its length is an even multiple of sixteen. Refer to pages 56-58 of the Extended Basic manual for a more detailed description of how to redefine characters.

```
CALL LINK("CHSIZE",width,height)
```

Used to specify the size of each character in pixels. The width must be a numeric value from 1 to 8. The height must be a numeric value from 1 to 12. The character size will determine the number of rows and columns that can fit on the screen. Once the character size is set, The Missing Link automatically uses characters of that size when displaying or inputting data.

Character patterns are used starting at the upper left hand corner. If the character size is less than 8 x 8 the extra pixels at the bottom or right hand side of the pattern will be ignored. If the character size is greater than 8 pixels high then the extra pixels at the bottom will be blank.

```
CALL LINK("FORMAT"[,format_code,number1,number2])
```

Used to determine the format used when displaying a number on the screen.

If the format code is "0" or if no numbers are passed to the subroutine then numbers will subsequently be displayed in standard Basic format.

If the format code is not zero then results will be similar to those obtained when using the IMAGE statement in Extended Basic. The number being displayed will always occupy a predetermined amount of space on the screen.

Number1 determines the number of characters to the left of the decimal point. Number2 specifies the number of characters to the right of the decimal point plus the decimal point. Thus, adding number1 to number2 will determine how many columns are required to print a number. When using scientific notation, add four characters if using a two digit exponent, or five characters if using a three digit exponent. If the number is too large, asterisks will be printed to identify the overflow condition. No error message will be issued.

The following format codes can be used:

0 - Standard Extended Basic format.

1 - Positive numbers will have a space displayed instead of a plus sign. If the number is long enough, an extra digit can be displayed instead of the plus sign.

2 - Positive numbers will have a space displayed instead of a plus sign.

4 - Both positive and negative numbers will have their signs displayed.

8 - Scientific notation with a two digit exponent. Positive numbers will have a space displayed instead of a plus sign.

12 - Scientific notation with a two digit exponent. Both positive and negative numbers will have their signs displayed.

24 - Scientific notation with a three digit exponent. Positive numbers will have a space displayed instead of a plus sign.

28 - Scientific notation with a three digit exponent. Both positive and negative numbers will have their signs displayed.

2.4.1. Changing The Default Printing Action

Several optional changes can be made to the default printing action by using the CALL LOAD subprogram. The following values may be useful:

CALL LOAD(11110,64,72) Blanks the background. (Default action)

CALL LOAD(11110,16,0) Leaves the background unchanged.

CALL LOAD(11112,224) Sets the pen to "pendown." (Default action)

CALL LOAD(11112,64) Sets the pen to "penerase."

CALL LOAD(11112,40) Sets the pen to "penreverse."

CALL LOAD(11080,2,129,24,0,22) The color of both the foreground and background pixels of the character pattern will be changed to the current penhue. This is the default action.

CALL LOAD(11080,16,0,209,92,19) Only the foreground pixels of the character pattern will be changed to the current penhue. The background pixels will remain unchanged.

2.5. Cartesion Graphics

These routines let the programmer plot points, lines, circles, and boxes on the screen. If the program is operating in the 16 color mode, they can also be used to change the pen color. If the window size is smaller than the full screen, then the graphics will only be displayed inside the window. The REVWIN subroutine can be used to specify that the graphics will only be displayed outside the window. No problems will result from drawing either partly or totally off the edges of the screen.

In all cases, both of the optional penhue values must be included to have any efect. The pen position used when generating turtle graphics has no effect when plotting cartesian graphics.

```
CALL LINK("PIXEL",row,column[,foreground_color,
        background _color])
```

This routine places a pixel on the screen. Including the optional color values will simultaneously change the penhue.

```
CALL LINK("LINE",row1,column1,row2,column2
        [,foreground_ color,background_color])
```

Draws a line between the points specified by row1,column1 and row2,column2. Including the optional color values will simultaneously change the penhue.

```
CALL LINK("CIRCLE",row,col,radius[,suppression_code,
        foreground_color,background_color])
```

Draws a circle of any radius with the center at the point specified by the row and column.

The circle is made up of eight arcs. Certain graphics applications may require that only part of the circle be displayed. Any combination of these eight segments can be blanked with the optional suppression code. This should be a number between 0 and 255. If the optional suppression code isn't passed, or if it is zero, then the entire circle will be displayed. Otherwise you can simply add up the numbers of the arcs you want to suppress and supply that number to the circle routine. *See page 18 for the arc supression code chart.*

Including the optional color values will simultaneously change the penhue. Be sure to supply a zero for the suppression code if you are changing the penhue and want to display the entire circle.


```
CALL LINK("BOX",row1,column1,row2,column2
        [,foreground_color,background_color])
```

Draws a rectangle between the points specified by row1,column1 and row2,column2. Including the optional color values will simultaneously change the penhue.

2.6. Turtle Graphics

The Missing Link gives the programmer many of the turtle graphics commands available in the Logo language with some improvements. Turtle graphics can now be in different colors, you can never run out of "ink", and the turtle has to travel 3 screen widths before it will "wrap around" the screen.

When The Missing Link is loaded, the turtle is placed at row 96, column 120. The heading is 0 degrees, which is straight up. The turtle is always hidden from view.

```
CALL LINK("TURN",angle)
```

This routine is used to turn the turtle. The angle is a number specifying the degrees the turtle is to turn. If the angle is negative, the turtle is turned to the left (counterclockwise); if positive it is turned to the right (clockwise). Angles larger than +180 degrees or less than- 180 degrees will "wrap around" (i.e. +380 degrees will become +20 degrees).

```
CALL LINK("FWD",distance[,angle,1])
```

Used to move the turtle forward the specified distance in pixels. If the distance is negative the turtle will be moved backwards. The distance can be a number from -32767 to +32768.

You can specify an optional angle, which causes the turtle to make a turn after completing its forward motion. Also, including a "1" for the optional third parameter will cause the turtle to return to the point it started from.

```
CALL LINK("PUTPEN",row,column[,angle])
```

Used to "pick up" the pen and move it to a specified location. The row should be a number from 0 to 192; the column can be from 0 to 240. If a zero is passed for either (or both) the row or column, the current value will not be changed.

Including the optional angle will cause the turtle heading to be set to the specified direction.

```
CALL LINK("GETPEN",row,column,angle)
```

Returns the current position of the turtle in the three numeric variables specified.

2.7. Sprite Graphics

The Missing Link can place up to 32 moving sprites on the screen at a time. Thirty-two different ASCII codes are available for sprite patterns. These ASCII codes are independent of the normal ASCII codes used when printing text. Operations can be performed simultaneously on consecutive numbered sprites, changing their locations, patterns, colors or motions at the same time.

```
CALL LINK("CHAR",ASCII_code,hexadecimal_string)
```

Used to define sprite patterns. With a few differences, this is the equivalent of the CALL CHAR subprogram in Extended Basic. There are no default sprite patterns. Your program has to define the patterns of any sprites that are used.

Only the ASCII characters from 1 to 32 can be used as sprite patterns. Double size sprites use four successive patterns and must start at ASCII 1,5,9,13,17,21,25, or 29.

The hexadecimal string that defines the sprite pattern can be up to 240 characters long. This means that up to 15 consecutive ASCII characters can be redefined each time this subprogram is called. When defining a series of sprite patterns, trying to define ASCII characters higher than 32 with a long hexadecimal string will result in the excess string being ignored. This means you cannot define both sprite patterns and character patterns in the same CALL LINK("CHAR") operation.

```
CALL LINK("SPRITE",sprite_#,ASCII[,color,row,col,
        row_velocity, col_velocity])
```

Used to create sprites, set them in motion, or modify any of their attributes. Notice that the parameters used by this routine should be provided in the same order used by the CALL SPRITE subprogram normally used in Extended Basic. However, the number sign (#) should not be placed before the sprite number.

If the sprite number is between 1 and 32 then only a single sprite will be created or modified. If the sprite number has three digits or more, then successive sprites will be created or modified simultaneously. For example, number 804 will simultaneously effect 8 sprites, starting with number 4. Number 2102 will effect 21 sprites starting with number 2. (Note that number 212 effects 2 sprites starting with number 12, not 21 sprites starting with 2.)

The ASCII, color, row, column, row velocity, and column velocity all operate as they do in Extended Basic's SPRITE subprogram. Although most of the parameters are optional, the ASCII, color, row, and column must be provided when first creating a sprite for it to be visible.

Once a sprite has been created, any of its attributes can be modified independently of the others. If the list of sprite attributes contains less than six values, the attributes that were omitted from the list will not be changed. Also, with

the exception of the velocities, providing a zero or a negative number for any attribute will result in no change to that attribute.

For example:

CALL LINK("SPRITE",1,0,0,96,120) will move sprite #1 to the center of the screen, but will not change the pattern, color, or velocities.

CALL LINK("SPRITE", 804,9,5) will effect 8 successive sprites, starting with #4. The eight sprites will now use ASCII 9 for a pattern, and will be a dark blue color.

CALL LINK("SPRITE",10,0,0,0,0,10) will give sprite #10 a row velocity of 10 without effecting any of the other current attributes.

Thus, it will be seen that the CALL LINK("SPRITE") subroutine can perform all the operations provided by Extended Basic's SPRITE, LOCATE, PATTERN, COLOR, and MOTION subprograms.

```
CALL LINK("DELSPR",sprite_number)
```

Used to delete either individual or consecutive sprites from the screen. The sprite number operates the same way as it does when creating sprites. See the description of CALL LINK("SPRITE") for more details on this.

If the sprite number is zero then all the sprites will be deleted.

```
CALL LINK("FREEZE")
```

Used to "turn off" the automatic sprite motion for all the sprites. Their motion will stop even if motions have been assigned using the CALL LINK("SPRITE") subroutine.

```
CALL LINK("THAW")
```

Used to "turn on" the automatic sprite motion for all the sprites.

```
CALL LINK("SPRPOS",sprite_#,row,column)
```

Used to retrieve the location of a sprite. The sprite number must be a number from 1 to 32. The location of the upper left hand corner of the sprite will be returned in the numeric variables used for row and column. These must be numeric variables or an error message will be issued.

```
CALL LINK("DSTNCE",sprite_#,sprite_#,numeric_variable)

CALL LINK("DSTNCE",sprite_#,row,col.,numeric_variable)
```

This functions identically to the CALL DISTANCE subprogram normally used in Extended Basic, except that the number sign (#) should not be used with

the sprite numbers. Please refer to page 80 of the Extended Basic manual for more details.

2.7.1. Checking For Sprite Coincidences

The CALL COINC subprogram is the usual method for determining coincidences in Extended Basic.

For example, in the program line below, X will be -1 if sprite #1 is within 10 pixels of sprite #2. Otherwise, X will be 0.

```
10 CALL COINC(#1,#2,10,X):: IF X=-1 THEN 100 ! COINCI-
DENCE HAS OCCURRED
```

But The Missing Link requires that the CALL LINK("DSTNCE") subroutine be used to determine coincidences.

In the program line below, X will be 100 or less if sprite #1 is within 10 pixels of sprite #2. Otherwise, X will be greater than 100.

```
10 CALL LINK("DSTNCE",1,2,X)::IF Xe THEN 100 ! COINCI-
DENCE HAS OCCURRED
```

```
CALL COINC(ALL,numeric_variable)
```

Extended Basic's CALL COINC subprogram can be used in the above manner to determine if any two sprites are in actual contact with each other. This is the only way The Missing Link can use the CALL COINC subprogram. Refer to page 64 of the Extended Basic manual for more details.

```
CALL MAGNIFY(magnification_factor)
```

Extended Basic's CALL MAGNIFY subprogram works in the normal manner. Refer to page 118 of the Extended Basic manual for more details.

2.8. Peripheral Access

The Missing Link can load and save pictures in standard TI-Artist format. A single density screen dump can be obtained at any time by calling a subroutine from within a program, or else by simply pressing the <FCTN> and <CTRL> keys at the same time.

When saving or loading pictures, there must be at least one disk file available that has not been opened by your Extended Basic program.

```
CALL LINK("LOADP",device_name[,1])
```

Used to load a screen from disk. The device name should be a string that specifies the disk number and the file name. An example of a valid device name is "DSK1.PICTURE".

If the filename is "DSK1.PICTURE", then The Missing Link will first search DSK1 for a file named "PICTURE_C". If that file is successfully found it will be loaded as the color data for the screen. Whether or not the color file is found, The Missing Link will then look for a file named "PICTURE_P". If that file is found it will be loaded as the picture data for the screen.

If The Missing Link is operating in the 2 color mode, then there will be no search for the color data, and the screen colors will not be changed. If The Missing Link is operating in the 16 color mode but fails to find the color data, the screen colors will be set to black on cyan. In either case, an I/O ERROR message will be issued if the picture data cannot be found.

When operating in the 16 color mode it is possible to suppress the search for the color data. Simply include the optional "1" after the device name to do so. The screen colors will remain unchanged when loading a picture in this manner.

```
CALL LINK("SAVEP",device_name[,1])
```

Used to save a screen to disk. The device name should be a string that specifies the disk number and the file name. An example of a valid device name is "DSK1.PICTURE".

If the filename is "DSK1.PICTURE", then The Missing Link will first save the color data to DSK1 in a file named "PICTURE_C", unless THE MISSING LINK is operating in the 2 color mode. The Missing Link will then save the picture data to DSK1 in a file named "PICTURE_P". If The Missing Link is unable to save the files to disk then an I/O ERROR message will be issued.

When operating in the 16 color mode, it is possible to suppress saving the color data. Simply include the optional "1" after the device name to do so.

Sprite data cannot be saved to disk.

```
CALL LINK("DUMP")
```

This subroutine is used to produce a single density graphics screen dump on Epson compatible printers. Pixels that are set to the foreground color will be black, while pixels set to the background color will be white. Sprites are not included in the screen dump. You can press Fctn 4 to break out of the screen dump. However, that will also cause your Extended Basic program to halt unless you have included the ON BREAK NEXT statement.

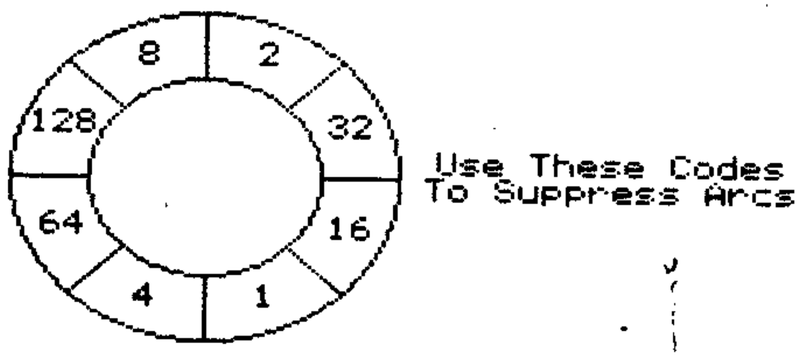
See page 18 for information on how to configure The Missing Link so the screen dump codes match your particular printers requirements.

Another way of obtaining a screen dump is to press the <Fctn> and the <Ctrl> keys simultaneously. Please note that this method is not functional on the Geneve 9640.

IMPORTANT! **If you accidentally hit the <Fctn> and the <Ctrl> keys and the printer is off, then your program will freeze until you either turn on the printer or else press <Fctn 4>.**

Supression Code Chart

Use the codes listed in each arc to supress that arc when drawing circles.
(See page 12 for information on drawing circles)



3. FONTS

3.1. Defining A Font Using CHARDEF

An Extended Basic utility program is included that makes it easy to modify existing fonts or create original fonts. The filename of the program is CHARDEF. The Missing Link must not be loaded when using CHARDEF.

To run the program, select Extended Basic from the master title screen. Then type RUN "DSKn.CHARDEF" and press <Enter>. CHARDEF will load and run.

The main menu gives you the choice of Editing characters, Saving the font, Loading a font, or Exiting. The on-screen prompts should be self explanatory. The main menu always shows the current font that is loaded, and this will be updated as you edit each character.

Fonts are saved and loaded in the standard INTERNAL, FIXED 241 format.

3.2. Loading A Font

The Missing Link includes five different fonts that can be loaded by your Extended Basic program. The filenames of the five fonts are: 46FONT, 48FONT, 57FONT, 68FONT, and 88FONT.

These files are in INTERNAL,FIXED 241 format. Each file is eight sectors long and contains seven hexadecimal strings of 240 characters. These strings can be read from disk and then used with the CALL LINK("CHAR") subroutine to change the character definitions.

The following lines of Extended Basic code demonstrate a subroutine that reads a font from disk and loads it into a program. This subroutine can be incorporated in your Extended Basic programs if you need to load a font.

```
10 A$="DSK1.68FONT"

20 GOSUB 100 ! go to sub that loads font

25 CALL LINK("CHSIZE",6,8) ! redefine character size
```

... program continues ...

```
100 OPEN #1:A$,VARIABLE 241,INPUT,INTERNAL :: FOR
C=33 TO 128 STEP 15 :: INPUT #1:A$ :: CALL
LINK("CHAR",C,A$) :: NEXT C :: CLOSE #1 :: RETURN
```

Two numbers in the front name specify the smallest character size that can be used with that font. Use of a larger character size will result in the same size characters being printed, but spaced further apart. Using too small a character size will result in partial characters being printed with no spaces between them. You will usually want to change character size when loading a font, as demonstrated in line 25 of the previous program.

4. CONFIGURING THE MISSING LINK

The Missing Link can be configured to match the specifications of your printer. This should be done if it is found that the graphics screen dump routine doesn't perform properly. If your printer is of the daisy wheel type, or if it doesn't support Epson graphics, the screen dump can not be made to work correctly.

Follow this procedure to configure The Missing Link. From the master title screen, select Extended Basic. Then place The Missing Link disk in disk drive #1 and do the following steps:

OLD DSK1.TML <Enter>

MERGE DSK1.CONFIG <Enter>

RUN <Enter>

Follow the on screen prompts as they appear. You will be asked for a printer device name. This can be "PIO" or "RS232" but it has to end in "CR". Then you will be asked for some special printer codes. You will have to look in your printer's instruction manual to find out this information. At one point you will be asked for "PRINTER CODES FOR SINGLE DENSITY GRAPHICS." Three codes are provided. If your printer only requires two codes, make sure the first code is a backspace (08), then change just the last two codes to match your printer's requirements. Finally, you will be asked for the code to reset the printer. If your printer doesn't have such a code, just press <Enter>.

After setting the printer specifications, you will be given the option of loading a font, which then becomes the default font. To load the font properly, it must be in the standard INTERNAL, FIXED 241 format. All the fonts on the program disk are in this format. Simply follow the instructions that appear on the screen.

Next you will be given the option of redefining the cursor used in the command mode. The hexadecimal patter provided as a default will define a Texas shaped cursor. You can redefine the cursor to you liking, or simply press <Enter> if th Texas cursor is desired.

Finally, you will be given instructions for saving your customized version of The Missing Link.

4.1. Saving Stack Space

Bit-mapped graphics require a lot of video memory. This memory is obtained at the expense of stack space, which is especially limited when using the 16 color mode.

It is important to realize that the reduced stack space does not decrease the maximum size that an Extended Basic program can be. Both the program and all numeric values generated by the program are contained inside the 32K memory expension. As before, there are 24488 bytes of free space available for a program. The only reduction is in stack space.

Extended Basic uses the stack for a number of purposes. After the prescan, it contains a list of all the variable names that are used by the program. Both string variable names and numeric variable names are in this list, as well as any array names. The stack space needed for each entry in the list is eight bytes plus the number of characters in the name. The prescan also generates a list of all the named subprograms such as JOYST, KEY, LINK and so on. User defined subprograms are also contained in this list. The stack space needed for each entry in this list is eight bytes plus the number of characters in the name. Finally, every string used by the program is also stored in the stack.

One way to conserve stack space is to limit your use of named subprograms. The stack space will not be significantly reduced if you use just a few named subprograms. However, if you are accustomed to writing a lot of your own subprograms, you should convert them to GOSUBs and ON GOSUBs wherever possible.

Stack space can be conserved by using as few numeric variables as possible, and by keeping their names as short as possible. Use numeric constants when possible, since constants require no entry in the list of variable names. Because the actual numeric values are stored in the 32K expansion, numeric arrays require only one entry in the list per array, so very little stack space is used.

Because the actual string is also stored in the stack, strings are the worst offender as far as using up stack space.

Instead of using string variables, use string constants whenever possible. Following are two examples that display text on the screen. The first example uses 28 bytes more stack space than the second:

```
10 A$="THIS IS A TEST"::CALL LINK("PRINT",1,1,A$) !
Uses more stack space

10 CALL LINK("PRINT",1,1,"THIS IS A TEST") ! Uses
less stack space
```

If you must use string variables, reuse the same variable name as many times as possible. Keep the number of string variables to a minimum. Following are two examples that redefine characters. Because the second example reuses A$, it uses 30 bytes less stack space than the first.


```
10 A$="FFFFFFFFFFFFFFFF" :: CALL LINK("CHAR",40,A$)

12 B$="FF818181818181FF" :: CALL LINK("CHAR",80,B$)
!Uses more stack space

10 A$="FFFFFFFFFFFFFFFF" :: CALL LINK("CHAR",40,A$)

12 A$="FF818181818181FF" :: CALL LINK("CHAR",80,A$)
!Uses less stack space
```

If possible, avoid string arrays, as even a small array is likely to cause a MEMORY FULL condition. Instead, keep strings in DATA statements and have your program READ them as needed. Following are two examples. The first reads strings from a DATA statement into a string array, where they are ready for use. The second uses 122 bytes less stack space by leaving the strings in the DATA statement until they are needed. The latter method does have the disadvantage of being slightly slower.

```
10 FOR I=1 TO 10  :: READ A$(I) :: NEXT I

20 CALL LINK("PRINT",1,1,A$(7)) ! Print String7 on
the screen

100 DATA String1,String2,String3,String4,String5,
String6,String7,String8,String9,String10

10 FOR I=1 TO 7 :: READ A$ :: NEXT I

20 CALL LINK("PRINT",1,1,A$) ! Print String7 on the
screen

100 DATA String1,String2,String3,String4,String5,
String6,String7,String8,String9,String10
```

Loading a font requires a lot of memory, which can cause a MEMORY FULL condition. If this happens, and if your program only needs to have one font loaded, try loading the font ahead of time. This can be done when you configure The Missing Link, or else by using a short program to load the font that then RUNs the main program.

Despite your best efforts, if you write a very long program, it may be impossible to conserve enough string space when using the 16 color mode. If that happens, you will have little choice but to go to the 2 color mode.

4.2. Converting Program Files To IV254 Files

Internal, Variable 254 is the format used by Extended Basic when saving or loading programs that are larger than the available stack space. Because of this, one of the side effects caused by The Missing Link's reduced stack

space is that all except very short Extended Basic programs must be saved to disk in INTERNAL, VARIABLE 254 format.

When The Missing Link has been loaded and you are writing a program, Extended Basic will automatically save it in IV254 format when necessary. However, there may be times when you want to load a program that was previously saved in PROGRAM format. If the program is too large, Extended Basic will issue the error message: I/O ERROR 02.

A utility has been included with The Missing Link package that makes it possible to convert already existing PROGRAM files into IV254 format. To convert from one format to the other:

1.. Select Extended Basic from the master title screen, then do the following:

2. Place The Missing Link disk in a disk drive. (drive number "n")

3. CALL INIT :: CALL LOAD("DSKn.CONVERT") <Enter>

4. CALL LINK("PROG") <Enter>

5. Place the disk containing the program format file in drive #n

6. OLD DSKn.PROGRAMNAME <Enter> - This will load the PROGRAM format file.

7. CALL LINK("IV254") <Enter>

8. SAVE DSKn.PROGRAMNAME <Enter> - This will save the program in IV254 format.

Repeat the process as many times as desired, starting with CALL LINK("PROG"). When finished using the CONVERT utility, always return to the master title screen.

5. PROGRAMMING EXAMPLES

To demonstrate the advanced capabilties of The Missing Link two Extended Basic program examples have been included on your Missing Link diskette. They are:

TMLDEMO - a menu driven program that demonstrates virtually every function of The Missing Link. To load The Missing Link demostration program follow these instructions:

Put The Missing Link disk into disk drive #1. Load and run The Missing Link (the TML program) first. Select one disk file and sixteen colors. Load and run the file TMLDEMO.

PS - PaperSaver; a utility program that lets you see precisely how text prepared with TI-Writer is going to look before it is printed. Instructions for using PaperSaver can be found in the following chapter.

These programs may be listed, printed and altered to suit your own needs. Both serve as fine examples of just what can be done with a standard TI-99/4a system, Extended Basic, and The Missing Link.

This page intentionally left blank.

6. PAPERSAVER

6.1. Introduction

PaperSaver is a utility program that, for the first time, lets the TI-99/4A user see precisely how text that has been prepared using TI-Writer is going to look before it is printed. The entire page is seen without resorting to the visually confusing practices of windowing or scrolling. Because each page can be viewed before printing, you will no longer waste reams of paper while perfecting the format of your documents.

Also, though not a word processor, PaperSaver will permit minor corrections to lines of text. This saves you from having to return to TI-Writer to repair minor mistakes.

PaperSaver requires The Missing Link in order to work properly. Although there are a few short assembly language subroutines that perform string handling, it is a remarkable fact that PaperSaver is written almost entirely in Extended Basic. PaperSaver is a good example of the powerful text and graphics capabilities that can be unleashed by teaming Extended Basic with The Missing Link.

6.2. Preparing The Text File

In order to use PaperSaver, a text file must first be created using TI-Writer. Prepare a document in the usual manner using the Editor program of TI-Writer. Include any of the "dot" commands that you need, such as .FI or .AD. You can also use any of the other formatter commands such as the "@@" for overstriking and the "&&" to underline words. Save this file to disk.

Load the Formatter. The Input Filename should be the file you just saved.

Next, instead of sending the formatted file to your printer, print it to disk by entering DSKn.FILENAME as the Print Device Name. Do not include .CR or .LF in the filename.

You will usually choose the next four default options: No mailing list; All the pages; One copy; No pause at end of page.

The formatted document will be saved to disk under the specified filename. This is the file that you will be previewing with PaperSaver.

There are several limitations that should be kept in mind when preparing a text file that will be viewed with PaperSaver. Because of memory and graphics limitations, PaperSaver cannot handle a width of more than 80 columns on a page. Therefore, if using .RM, make sure the right margin is set to less than 80. PaperSaver can only display text. Including graphics data will cause unpredictable results on the screen. You can safely include printer codes in

your document that will select expanded print, subscripts, backspace, etc. However, they will be ignored by PaperSaver.

6.3. Loading PaperSaver

Follow this procedure to load PaperSaver:

1. From the master title screen, select Extended Basic.

2. Place The Missing Link disk into drive number 1

3. Type: RUN "DSK1.TML" <Enter>

4. Select 1 disk file and the 16 color mode

5. Type: RUN "DSKn.PS" <Enter>

PaperSaver will load and run.

6.4. Using PaperSaver

PaperSaver first displays the title screen and loads a character font. It then displays the main screen. The Missing Link disk can be removed once the main screen is displayed.

Next, you are asked for the name of your printer. This should be the same name you usually input when using the formatter portion of TI-Writer. Type in the name, then press <Enter>. (The default printer name can be changed in program line 280.)

Then you are asked for the name of the text file that you have previously formatted. Place the disk containing this file into a disk drive, type in the filename, then press <Enter>. If PaperSaver cannot find the file in the specified drive, it will issue an error message, then request the filename again.

Once the printer name and file name have been entered, you are presented with a menu containing four choices: View File; Print File; New File; Exit. Press the key that corresponds to your choice.

6.5. View File

This is the menu choice that you will use most frequently. After making this choice, you will be asked for a page number. Type any page number, then press <Enter>.

At this point, PaperSaver will scan through the disk file looking for the specified page. If you should specify a page number that is higher than the last page number contained in the file, then "EOF" (for End Of File) will be displayed next to the page number of the last page in your document. Press any key to return to the menu.

If the page was found, then a miniature replica of the page is created in the left hand section of the screen. This looks exactly as the page will look when it is printed. The page is light green, and the characters are dark green. Underlines are shown in dark green, and characters that will be overstruck are shown in black.

Pointers to the current line appear on each side of the page, and the line number is displayed. The text contained in this line is then printed in the lower right corner of the screen. Underlining is clearly show, and overstruck characters are displayed in black. Finally, a short menu of options is displayed.

Use the up and down arrow keys (Fctn E and Fctn X) to move the line pointers up and down on the page.

Press "P" when you are ready to input a new page number. The default page number increments to the next page, except when you are viewing the last page. Type the desired page number, then press <Enter>.

Press "L" if you want to go directly to a line rather than use the cursor keys to get to that line. Type the desired line number, then press <Enter>.

Press "E" if you want to edit a line. An area clears where the line can be modified, then the text contained in the current line (minus underlines and overstrikes) appears in this cleared area. The following function keys can be used as indicated: Fctn S (left cursor), Fctn D (right cursor), Fctn 1 (delete), Fctn 2 (insert), Fctn 3 (erase from cursor on). Modify the text as desired, then press <Enter>.

Next, you are asked if you want to underline any characters. Press "Y" or "N" to indicate your choice. If you press "Y", the line you just edited appears in the input area. Replace any letters that are to be underlined with underlines. You don't have to erase those characters that will not be underlined. Press <Enter> when finished.

Next, you are asked if you want words to be overstruck so they will be printed in boldface. Press "Y" or "N" to indicate your choice. If you press "Y", the line you are editing (minus any underlines) appears in the input area. Use the space bar to erase any characters that are not to be overstruck. Leave only those characters that you want in boldface. Press <Enter> when finished.

The line you have edited will then be updated in the left hand portion of the screen, the menu choices will reappear, and the text contained in the line will appear in the lower right hand corner of the screen.

IMPORTANT! Editing a line DOES NOT result in any changes to the disk file. If you want to change the text permanently, you will have to return to TI-Writer to make the changes.

Press "C" to print out the page as it is currently displayed. Be sure your printer is turned on and that it is on line.

Press "Fctn 9" to return to the main menu.

6.6. Print File

If you have viewed all the pages of the disk file and found them to be correct, then the file can be printed without returning to TI-Writer. Press "P" to print the file out. You are given the option of having a pause at the end of each page. Choosing this option lets you feed single sheets of paper to your printer. If you have continuous paper, simply press "N".

All the pages in the file will be printed out. Be sure your printer is turned on and is on line.

6.7. New File

Press "N" if you want to change either the printer name or the disk file name.

6.8. Exit

Press "E" when ready to leave PaperSaver.

LIMITED WARRANTY

Texaments extends no warranty for The Missing Link or PaperSaver beyond the physical part consisting of the data diskette and documentation. This limited warranty does not extend to the programs and data files contained in the software media and described in the documentation. The user takes sole responsibility for the use of, improper use of, and loss of use, of the programs and data files contained in the software media.

The data diskette included with this package are warranted to be free from defects in material and workmanship under normal use and conditions for a period of 90 days from date of delivery to the original purchaser. All defective diskettes will be replaced free of charge when returned within the 90 day warranty period postage prepaid to Texaments. After the warranty period expires all replacements will require a fee of $5.00 to accompany the original diskette(s) and a valid sales receipt.

Texaments
53 Center Street
Patchogue, New York 11772

WANTED!
EXTENDED BASIC PROGRAMS THAT USE THE MISSING LINK

If you have written an Extended Basic program that uses the advanced capabilities of The Missing Link, and would like to distribute it to others worldwide, send it to Texaments. Please include your full address and telephone number with all correspondence and program submissions. Programs on disk should be sent to:

Texaments
Attn: Missing Link Programs
53 Center Street
Patchogue, New York 11772