14-Feb-90 Rev. 7

99/9640 FORTRAN Manual Errata Sheet

The following are a list of errors and extensions to the 4.3 99/9640 FORTRAN manual which make up version 4.4:

pp 01-3, Section 1.1 Getting Started:

Replace paragraph starting with "If you have a hard ..." with:

If you have a hard disk using the MYARC HFDC, you may wish to place the TI-99 implementation of the compiler in the directory:

HDS1.DSK.FORTCOMP

pp 02-1, Section 2.1 TI-99 (GPL Implementation) Editor

Add the following entry to the editor menu selection list:

7 Exit

pp 02-5, Section 2.1.7 Exiting the Editor

Add the following statements:

The editor can also be exited by selecting item 7 (Exit) from the Editor Menu Display.

The prompt "*Are You Sure (Y/N)*" is only displayed if the working buffer contains a program which has been modified and not saved. For example, if you load a program into the editor and do not modify it, the editor will exit immediately upon depression of the "7 (Exit)" or "Fctn/Back".

pp 03-2, Section 3.1.2 Statement Format

FORTRAN statements now extend from columns 7 to 80, not 7 to 72. The columns 73 to 80 are no longer ignored.

pp 03-2, Section 3.1.3 Constants

Add to the section "A. Numeric Constants":

6. Complex Constants
7. Double Precision Complex Constants

pp 03-6, Add two new sections 3.1.3.9 and 3.1.3.10

3.1.3.9 Complex Constants

Complex constants have two single precision (real *8) components, a real component and an imaginary component. These real and imaginary components are specified as two single precision numbers, enclosed in parenthesis, and seperated by a comma:

(real, imaginary)

The following are examples of valid complex constants:

```
( 1.0, 2.0 )
( 1.0, -2.0 )
( 0.0, 3.14159 )
```

A complex constant requires *eight* bytes of storage.

3.1.3.10 Double Precision Complex Constants

Double precision complex constants have two double precision (real *8) components, a real component and an imaginary component. These real and imaginary components are specified as two double precision real numbers, enclosed in parenthesis, and seperated by a comma:

(real, imaginary)

The following are examples of valid complex constants:

```
( 1.0d0, 2.0d0 )
( 1.0d0, -2.0d0 )
( 0.0d0, 3.14159d0 )
```

A double precision complex constant requires *sixteen* bytes of storage.

If one of the components is specified as single precision real, and the other as double precision, then the resulting constant will be of type double precision complex. Integer numbers within the const are not allowed:

(1, 2.0d0)

is an illegal complex constant since it contains an integer value (1).

pp 03-7, Section 3.1.5 FORTRAN Variables

Change the last sentence of the section starting ... Array variables must be declared... to:

Array variables must be declared using the INTEGER, REAL, DOUBLE PRECISION, LOGICAL, CHARACTER, COMPLEX, COMMON, or DIMENSION statements.

pp 03-11, Section 3.2.1 Arithmetic Assignment Statement

Change the definition of **a** to:

a is a variable (scalar or array) whose type is *integer, real, double precision, complex* or *character*, and

pp 03-19, Section 3.3

Add the following to the allowable I/O Statements:

- **ACCEPT** Statements
- **PRINT** Statements
- **TYPE** Statements

pp 03-19, Section 3.3.1 READ Statement

Add the following General Form:

read (i, n [,keys]) variable list
or
read [n],variable list

Add the following keyword:

IOSTAT=m where m is an integer scalar variable into which status information is to be stored if an error occurs (same as STATUS=).

pp 03-21, Section 3.3.3 Variable Lists

Add the following statements to the end of section 3.3.3:

A *variable list* may contain an embedded string, enclosed in apostrophes per normal FORTRAN string conventions. The corresponding FORMAT item for the string MUST be the FORMAT specifier **A** (with no length), or the length must match the string length.

For example:

```
      WRITE ( 6, 9100 ) 'THE VALUE OF Z IS',Z
9100  FORMAT ( 1X, A, 1X, F8.2 )
```

This is especially useful for list directed formatting, i.e.:

```
PRINT ,'THE VALUE OF Z IS ',Z
```

is a valid output statement and will cause the following line to be output to the CRT:

THE VALUE OF Z IS 29.7123457

pp 03-22, Section 3.3.5, 3.3.6, 3.3.7 New Sections

Add the following sections:

3.3.5 ACCEPT Statement

The **ACCEPT** statement is similar to the READ statement, except that the unit number is always assumed to be unit 6, the CRT device.

General Form:

ACCEPT [n],variable list

where:

n is the label of a FORMAT statement, the name of an array which contains the FORMAT specification, or is omitted if ist directed formatting is used.

variable list is a normal I/O variable list to be transferred.

For example:

```
      ACCEPT ,X,I
      ACCEPT 9100,X,I
9100 FORMAT ( F10.2, I6 )
```

In the first example, the values for X and I will be solicited at the CRT device (console) using list directed formatting. In the second example, the values for X and I will also be prompted at the CRT, and FORMAT statement 9100 will be used to decode the data.

3.3.6 TYPE and PRINT Statements

The **TYPE** and **PRINT** statements are similar to the WRITE statement, but the output device is always the CRT or console device.

General Form:

```
TYPE  [n],variable list
PRINT [n],variable list
```

where **n** and **variable list** are the same as described for the **ACCEPT** statement.

For example:

```
      PRINT,I,J,X
      TYPE 9100,I,J,X
9100 FORMAT ( I6, I4, F10.2 )
```

would specify in the first example that I, J, and X are to be written to the CRT device using list directed formatting; whereas in the second example I, J, and X are to be tranferred to the CRT device according to FORMAT statement 9100.

3.3.7 List Directed Formatting

List directed formatting allows you to specify a read or write operation without specifying a FORMAT statement. This is mainly useful for debug statements or for output whose format is not important.

List directed formatting is activated by either omitting the FORMAT statement number or specifying an asterick for the FORMAT statment number. For example:

```
WRITE ( 6, * ) X, Z, (ARRAY(I),I=1,10)
READ  ( 6, * ) X, 'VALUE Z IS ',Z, (ARRAY(I),I=1,10)

TYPE *,A
PRINT ,Z
ACCEPT *,A,B,TI_MODE
```

Note in each example the FORMAT number is either omitted or replaced by an *asterick,* indicating list directed formatting.

List directed formatting assumes that the carriage control for the line is single carriage return/line feed (the space specifier), and assumes the following formats for each variable type:

I

INTEGER *1	I5
INTEGER *2	I8
INTEGER *4	I13

COMPLEX *8	2F16.7
COMPLEX *16	2F26.16
LOGICAL *2	L5

pp 03-27 A Format (Alphanumeric) rAw

Add the following before the example:

If **w** is omitted, then the width of the character string assumes the width of the variable list item, as follows:

INTEGER *1	1 character
INTEGER *2	2 characters
INTEGER *4	4 characters
REAL *4	4 characters
REAL *8	8 characters
CHARACTER	1 character
'string'	length of string

pp 03-38 3.4.7 Explicit Type Declarations

Change the statement ... *type* is one of the four declarations ... to:

where: *type* is one of the following six declarations:

INTEGER
REAL
DOUBLE PRECISION
LOGICAL
COMPLEX
CHARACTER

Add the following after the statement ... LOGICAL *2 - two bytes (default...:

COMPLEX *8	- eight bytes (same as COMPLEX)
COMPLEX *16	- sixteen bytes
CHARACTER	- one byte (same as INTEGER *1)

pp 03-39, Section 3.4.8 IMPLICIT Statement

Change the statement ... *type* is one of the four declarations... to:

type is one of the six declarations:

INTEGER
REAL
DOUBLE PRECISION
LOGICAL
COMPLEX
CHARACTER

Add the following after the statement ... LOGICAL *2 - two bytes ... :

COMPLEX *8	- eight bytes (same as COMPLEX)
COMPLEX *16	- sixteen bytes
CHARACTER	- one byte (same as INTEGER *1)

pp 03-45 FUNCTION Subprogram

Change the statement .. *type* is optional and is one of the specifications... to:

type is optional and one of the specifications INTEGER, REAL, DOUBLE PRECISION, LOGICAL, COMPLEX, or CHARACTER

Add the following after the line ... LOGICAL *2...:

COMPLEX *8
COMPLEX *16

pp 03-48, Section 3.5.6 Library Subprograms

Change the statement ... The subprograms are included ... to:

The subprograms are included in four supplied libraries, the FL (FORTRAN) library, the GL (Graphics) Library, the ML (Math) Library, and the CL (Complex Math) Library.

pp 03-49, Section 3.6.1 Program Statement

Add the following note at the end of the section:

The program name MUST be specified if using the symbolic debugger. It also MUST be the first statement of the program (i.e. not preceeded by comment lines) for the debugger to work properly.

pp 04-5, Section 4.4 Allocation Map

Change the table of letters vs. allocation types in COMMON area and in the Local Data Area to the following:

k - INTEGER *1	i - INTEGER *2
j - INTEGER *4	r - REAL *4
d - REAL *8	c - COMPLEX *8
e - COMPLEX *16	l - LOGICAL *2

pp 05-6, Section 5.1.5 Linker Errors

Change the statement under "IMO Error" starting with ... Possible causes include ... to:

The cause is a non-object (source or other) module being loaded.

pp 06-10, Section 6.5.3 Remove/Add Breakpoints

Add the following at the end of the "Breakpoint Execution" section:

In the MDOS version of the debugger, whenever a breakpoint is encountered, the first 64 bytes of the program area (locations 0 to 3f) are checked. These locations are used by MDOS, if your program should overwrite any of these locations (typically location 0), then errant program execution including lockups of the GENEVE computer could occur.

**Critical Location @xxxx has been corrupted;
Incorrect Value = aaaa, should be = bbbb.
Shall I fix location (Y/N)?

Answer **Y** and enter if you want the debugger to correct the location. If your application program actually modifies the location, then answer **N** and enter, and the debugger will assume from this point on that the modified value is the correct value.

pp 06-16 Inspect or Change WP, PC, or SR

Add the following statement to the end of the section:

In the **MDOS** implementation, the Program Counter (PC) display shows the current module, program line number, and FORTRAN label, as follows:

PC = hloc module, %line, *label

For example;

PC = A362 TESTPROGA, % 64, * 1200

would specify that the current breakpoint is at hexadecimal location A362, in module TESTPROGA, at line number 64, and FORTRAN label 1200.

pp 07-01, Section 7.0 Introduction

Following statement in paragraph 7 on page 7-1 which starts out with:

Therefore it is most efficient to reference the FL library FIRST...

should read:

Therefore it is most efficient to reference the ML library FIRST...

pp 07-2, Section 7.0 Introduction

Modify the statement ... Three libraries are supplied... to:

Four libraries are supplied for your use. They are:

FL - Main FORTRAN library
ML - Single and Double Precision Math Library
CL - Complex Math Library
GL - Graphics Library

Change the last sentence in the **ML** library description as follows:

Therefore you **MUST** reference the **ML** library FIRST, followed by the **FL** library.

Add the following paragraph describing the **CL** library:

The **CL** library contains COMPLEX *8 and COMPLEX *16 mathematical functions, such as CSIN, CCOS, CLOG, CDSIN, CDCOS, CDLOG, etc. You will only need to reference this library if you make reference to one of these higher math routines. Note that the **CL** library makes reference to several routines in the **ML** and **FL** libraries. Therefore you MUST reference the **CL** library FIRST, followed by the **ML** library and then the **FL** library.

pp 07-3, Section 7.1.1:

Add note at end of section 7.1.1:

When file names are interactively entered from the CRT, then the end of the list is indicated by typing the line >EOD. For example, given a command file of CRT, the following would input files HDS1.OBJ4, DSK2.CSIN:

```
HDS1.OBJ4
DSK2.CSIN
>EOD
```

pp 07-10, Section 7.2:

Add the following complex intrinsic library functions:

General Function	Routine Name	Definition	No. Arguments	Argument Type	Result Type	Inline?
Complex Type Conversions	CMPLX DCMPLX	y = a + jb (Real to Cmplx)	2	R D	C E	
	CONJG DCONJG	y = a - jb (Cmplx Conjucate)	1	C E	C E	
	REAL DREAL	y = a + j0 (Real Part)	1	C E	R D	
	AIMAG DIMAG	y = 0 + jb (Imaginary Part)	1	C E	R D	
Complex Abs Val	CABS CDABS	y=abs(x)	1	C E	R D	
Complex Sq Root	CSQRT CDSQRT	y=x**0.5	1	C E	C E	
Complex Logarithm	CLOG CDLOG	y=ln(x)	1	C E	C E	
Complex Trigono-metric Functions	CSIN CDSIN CCOS CDCOS	sine(x) cosine(x)	1	C E C E	C E C E	
Complex Exponen-tial	CEXP CDEXP	y=exp(x)	1	C E	C E	
Trigono-metric Functions	CDTAN CARSIN CDASIN CARCOS CDACOS CATAN CDATAN	arc-sine(x) arc-cosine(x) arc-tangent(x)	1 1 1	E C E C E C E	E C E C E C E	
Complex Hyperbolic Functions	CSINH CDSINH CCOSH CDCOSH CTANH CDTANH	hyperbolic-sine (x) hyperbolic-cosine(x) hyperbolic-tangent(x)	1 1 1	C E C E C E	C E C E C E	
	CASINH CDASNH CACOSH CDACSH CATANH	hyperbolic-arc sine(x) hyperbolic-arc cosine(x) hyperbolic-	1 1 1	C E C E C	C E C E C	

pp 07-14, section 7.3.5 CALL BREAD/BWRITE (MDOS Only)

The latest MDOS documentation indicates the file information header is 20 bytes long, not 18 bytes. Change the statement:

> It should be at least 18 bytes long....

to:

> It should be at least 20 bytes long...

(under no_sectors). Also change the example to:

```
integer *1 filehdr(20), filename(10)
data filename / 10htestfile     /
call bread ( filename, 0, 0, filehdr, error )
```

pp 07-21, section 7.4.13 CALL CMDSTR

Change the calling sequence to the following:

Calling Sequence:

> CALL CMDSTR (array [,ierror])

where:

> *array* is an integer *1 array of which the first byte is the length-1 of the array, and the remaining bytes will contain the command line string on return,
>
> *ierror* is an optional integer *2 variable which will be set to zero if the command was returned into the array, it will be set to -1 if the command line exceeded the length of the array.

pp 07-34; sections 7.3.6 and 7.3.7

Add the following new sections describing new utilities for MDOS:

7.3.6 CALL FORMAT (MDOS Only)

The FORMAT subroutine is called to FORMAT a floppy disk. It calls the MDOS utility to FORMAT the number of sectors specified.

calling sequence:

> call format (file, tracks, skew, interlace, density, sides, sectors, error)

where:

> file : is an array which contains the name of the device to format (e.g. DSK1.)
>
> tracks : is a one word integer variable which is the number of tracks per side to format (usually 40 or 80)
>
> skew : is a one word integer variable which is the desired skew setting (e.g. 2)
>
> interlace: is a one word integer variable which is the desired interlace (e.g. 1 for 1:1 i

density : is a one word integer variable which is the desired disk density (e.g. 2 for double density)

sides : is a one word integer variable which is the number of sides to format (1 for single, 2 for double)

sectors : is a one word integer variable which contains the number of formatted sectors

error : is a one word integer variable which will be set to zero if there is no error formatting the floppy, if non-zero an error occurred.

For example, the following will format the floppy disk in drive DSK1., for 40 tracks, a skew of 2, an interlace of 1, single density, double sided. It prints the number of formatted sectors when done.

```
call format ( 'DSK1. ', 40, 2, 1, 1, 2, isectors, ierror )
print *,isectors,' formatted, error is ',ierror
```

Note that after the floppy disk has been formatted, the first sector requires initialization of the floppy name and other information; and the second sector requires to be zeroed before the floppy disk can be used.

7.3.7 CALL CREATD (MDOS Only)

The CREATD routine is used to create a directory on the hard disk (or extended floppy disk) device specified.

calling sequence:

```
CALL CREATD ( directory, error )
```

where:

directory : is an array which contains the directory specification. It MUST be terminated with a period.

error : is a one-word integer variable which will contain any error returned.

For example, the following will create the directory 'TEMPA' on the hard disk HDS1:

```
character directory(12)
data directory / 12HHDS1.TEMPA. /
call creatd ( directory, ierror )
print *,'directory created, error=',ierror
```

pp 07-22, Section 7.5.1

Add the following to the discussion of character value, sprite color, dot row, and dot column:

character value is a one word integer variable or constant which contains the character number associated with the sprite. This number can vary from 128 to 255 (in TI-99/4A GPL mode) or from 0 to 255 (in MDOS mode).

sprite color has two different meanings depending on whether you are using sprite mode 1

color, from 1 to 16. In sprite mode 2, the *sprite color* is an eight word array, each word contains a number from 1 to 16 defining the sprite color for each horizontal line in the sprite.

dot row and *dot column* are one word integer variables or constants which specify the starting row and column of the sprite. *Dot row* can vary from 1 to 192 (top and bottom) in TI-99/4A GPL mode or from 1 to 208 in MDOS modes 2, 3 and 4 (graphics modes multicolor, 1 and 2; or sprite mode 1) to 216 in MDOS mode using modes 5, 6, 7, 8 or 9 (sprite mode 2). *Dot column* can vary from 1 to 255 (left to right) regardless of graphics mode.

Correct the first example as follows:

```
call spchar ( 128, z'ffffffffffffffff' )
call sprite  ( 1, 128, 5, 2, 1, 24 )
```

Add the following example of sprite mode 2:

```
      integer colors(8)
      data colors /  1, 3, 5, 7, 9, 11, 13, 15 /
c
      call setmod(9)
      call clear
      call spchar ( 128, z'f0f0f0f0f0f0f0f0' )
      call sprite ( 1, 128, colors, 2, 1, 24 )
```

The above example sets the current screen mode to 9 (graphics mode 7), clears the screen, defines a sprite character number 128, and defines a sprite using that sprite character and the colors defined in the "colors" array.

pp 07-23, Section 7.5.2:

Add the following to the discussion of character code in CALL SPCHAR:

character code is a one word integer variable or constant which contains the sprite character number to modify, from 128 to 255 (in TI-99/4A GPL mode) or 0 to 255 (in MDOS mode).

pp 07-35, Section 7.9.3:

Add the following text to the discussion of IRAND:

The random number returned can be randomized by calling the CALL RANDOM subroutine. For example,, the following will produce a set of non-repeating numbers upon each program execution:

```
      CALL RANDOM
      DO 100 I=1,100
        Y = IRAND ( 10 )
        PRINT *,Y
100   CONTINUE
```

pp 07-39, Section 7.10.1

Change the statement:

from 0 to 3 (TI-99 GPL mode), as follows:

to:

from 0 to 4 (TI-99 GPL mode), as follows:

Also, add the notes concerning TI-99/4A mode 4 usage (BITMAP mode):

When using CALL SETMOD(4) in TI-99/4A mode, you must precede the CALL SETMOD(4) statement with a CALL FILES(1) statement. This is because VDP RAM is at a premium when using SETMOD(4). Also, it is not possible to use the debugger when using bitmap mode 4, and text display output is possible, but normal screen scroll functions are not.

pp 07-44,Section 7.10.10

Remove notation for MDOS ONLY. CALL SETPIX is now available with the TI-99 and mode 4 (graphics mode 2).

Also, in graphics mode 2 under TI-99/4A, the color is specified as a number from 1 to 16, where 1 is transparent and 16 is white.

pp 07-45, Section 7.10.11

Correct section numbering (from 7.9.11 to 7.10.11). Also, remove notation that this routine is available under MDOS only; it is also available in TI-99/4A graphics mode 2 (set mode 4).

pp 07-45, Section 7.10.12

Remove notation for MDOS ONLY. CALL SETVEC is now available with the TI-99 and set mode 4 (graphics mode 2).

Also, in graphics mode 2 under TI-99/4A, the color is specified as a number from 1 to 16, where 1 is transparent and 16 is white.

pp 07-47, Section 7.10.14:

Corrected Calling Sequences:

```
CALL HBLKMV (row_uls, col_uls, row_uld, col_uld, norow, nocol, color )
CALL HBLKCP (row_uls, col_uls, row_uld, col_uld,  norow, nocol )
CALL LBLKMV (row_uls, col_uls, row_uld, col_uld,  norow, nocol, color, logic )
CALL LBLKCP (row_uls, col_uls, row_uld, col_uld,  norow, nocol, logic )
```

In addition, the valid values for "*logic*" are:

```
0   IMP    DC = SC
1   AND    DC = SC * DC
2   OR     DC = SC + DC
3   EOR    DC = !SC * DC + SC * !DC
8   TIMP   if SC=0 then DC=DC else DC=SC
9   TIAND  if SC=0 then DC=DC else DC=SC*DC
10  TOR    if SC=0 then DC=DC else DC=SC+DC
11  TEOR   if SC=0 then DC=DC else !SC*DC+SC*!DC
12  TNOT   if SC=0 then DC=DC else DC=!SC
```

where:

DC = Destination Color

pp 07-57, section 7.12.2

The calling sequence for MALLOC is incorrect. The corrected sequence is:

CALL MALLOC (nopages, spage, speed, error, noapages, nofpages)

pp 07-59, section 7.13 Extended Graphics Library

Add the following new section to the manual:

7.13 Extended Graphics Library

The following routines are available through Public Domain as an extended graphics library of functions.

Two routines are currently provided, a CIRCLE drawing subroutine originally coded by J. Syzdek (for assembly), and two subroutines to move pixel data between CPU and Video RAM.

7.13.1 CALL CIRCLE

CALL CIRCLE plots a circle on the screen, given the center x coordinate, the center y coordinate, the radius (in pixels), and the color code. This routine is available under MDOS and also under TI-99/4A set mode 4 (graphics mode 2).

Calling Sequence:

MDOS Form:
CALL CIRCLE (center_x, center_y, radius, color)

TI-99/4A Form(graphics mode 2):
CALL CIRCLE (center_x, center_y, radius [,fore_color] [,back_color])

where:

center_x : is a one word integer variable which is the x pixel coordinate,
center_y : is a one word integer variable which is the y pixel coordinate,
radius : is a one word integer variable which is the radius of the circle in pixels
color : is a one word integer variable which is the color code to make the circle.
fore_color : is a one word integer varaible which is the foreground color for the circle (TI-99/4A mode only), a number from 1 to 16.
back_color : is a one word integer variable which is the background color for the circle (TI-99/4A mode only), a number from 1 to 16.

For example, the statements:

```
call setmod ( 9 )
call circle ( 25, 28, 20, 7 )
```

will set the screen mode to 9 (graphics mode 7) and create a blue circle at coordinates 25,28 of radius 20 on the upper left hand corner of the screen in MDOS mode. An equivalent for TI-99/4A mode might be:

```
call files(1)
call setmod(4)
call circle ( 25, 28, 20, 5, 2 )      ! blue on black background
```

7.13.2 CALL LMMC/CALL LMCM (MDOS Only)

The LMMC and LMCM routines perform logical moves of pixels from Video RAM (VRAM) memory CPU Memory (LMCM), and from CPU Memory to Video Memory (LMMC).

Calling Sequence:

```
CALL LMCM ( x_coord, y_coord, dots_x, dots_y, direc_x, direc_y, array, bytes_xfered )
CALL LMMC ( x_coord, y_coord, dots_x, dots_y, direc_x, direc_y, array, bytes_xfered[, logic] )
```

where:

x_coord: is a one word integer variable which specifies the starting pixel coordinate, from 0 to 511,

y_coord: is a one word integer variable which specifies the starting pixel coordinate, from 0 t to 1023,

dots_x : is a one word integer variable which is a value from 0 to 511, and is the # of pixels in the x direction comprising the window,

dots_y : is a one word integer variable which is a value from 0 to 1023, and is the # of pixels in the y direction comprising the window,

direc_x: is a one word integer variable, and is set to 0 for transfer in the right direction, and 1 for transfer in the left direction,

direc_y: is a one word integer variable, and is set to 0 for transfers in the down direction, a 1 for transfers in the up direction,

array : is a FORTRAN array that will contain the pixel values,

bytes_xferred : is a one word integer variable which will contain the number of bytes of data actually transferred, and

logic : is a one word integer variable which is the logic operation to perform (0, just copy) (optional)

This routine is only callable in graphics modes 4, 5, 6, or 7. The upon execution of the routine CALL LMCM, the array will contain a packed array of color pixel values, packed into bytes according to the graphics mode, as follows:

```
graphics mode 4,6  :  two pixels/byte
graphics mode 5:      four pixels/byte
graphics mode 7:      one pixel/byte
```

For example, the following will transfer an array of 10,000 pixel elements from a window at coordinates 50,80, and of size 200,100.

```
integer *1 pixels
common pixels(10000)
call setmod(8)                    ! graphics mode 6
call lmcm(50,80,200,100,0,0,pixels,notrans)
```

The following statement will transfer the pixel array back to the video ram screen:

```
call lmmc(50,80,200,100,0,0,pixels,notrans)
```

The utilities package has been extended considerably over the description in the manual. Please refer to the following rewritten utilities documentation:

9.0 UTILITIES (TI-99 GPL ONLY)

The UTILITIES program (item 9 on the 99 FORTRAN menu) provides several functions which enhance the usage of 99 FORTRAN, including:

a. A preferences package, which allows you to modify 99 FORTRAN parameters to your individual tastes,

b. A method to do a disk directory without exiting the 99 FORTRAN environment,

c. A transform utility which transforms downloaded files in display/fixed/128 format to display/fixed/80 or display/variable/80,

d. A method to save the modified preferences to disk (create new boot image) and to create a standalone editor/assembler option 5 (Load & Run) program from your FORTRAN program.

When item 9 is selected on the FORTRAN main menu, then four files are loaded into main memory and the following menu is displayed:

99 Utilities V4.41

Press:

1 To Modify Preferences
2 Do Disk Directory
3 Transform a File
4 Create New Menu
5 Exit

Press the number key associated with the function you wish to perform.

9.1 Modify Preferences

Item 1 on the Utilities MENU allows you to modify various parameters in the 99 FORTRAN MENU to suit your configuration or individual tastes. When this item is selected, the following screen is displayed:

Preferences

>Reload CharSet on Reset.	Y
Default # Files to Open.	3
Number of Lines/Page ...	56
32/40/80 Columns	40
Count for Cursor Blink .	200
Memory Model	E/A
Background Color	LIG BLUE
Foreground Color	WHITE
Character for Cursor ...	5F
Default Label for Printer	9
Terminating Printer Char	0A
Wild Card Label Binding	6

BOOT Disk Name: DSK.FORTCOMP.
Printer:

and the cursor will be positioned to the "Reload CharSet" column. To change a setting, merely enter the new number. To leave the setting alone, then just press enter to skip to the next entry.

9.1.1 Reload CharSet on Reset

This entry defines whether the MENU program will reload the standard character set in GPL on startup and every program exit. You may not want MENU to reload if you are running with a custom character set.

Also, if you answer N, then all access to GPL from 99 FORTRAN are bypassed (possibly useful if you are using a substitute GROMenvironment, and also useful if you want to run this program under MDOS using the EXEC program).

Press the "1" key to toggle the entry from Y to N (Yes to No).

9.1.2 Default # Files to Open

This is the default number of disk files to open. Setting this number higher will allow your FORTRAN program to open more disk files, without calling the CALL FILES subroutine.

Enter a number from 1 to 9, or ENTER to skip.

9.1.3 Number of Lines/Page

This parameter tells the compiler and the EDITOR how many lines there are per printer page. The default is 56, which is normal for 6 lines/inch, 8 1/2" by 11" paper.

9.1.4 32 or 40 or 80 Column Default

This parameter defines how many columns there are per screen line. The normal is 40. Usage of 80 Columns requires either a Mechtronics, DIGIT, 80-column card or a MYARC GENEVE. The default is 40.

Press key "1" to toggle to the next selection (32/40/80). CAUTION: Do not leave this parameter in 80-column mode if you do not have 80-column hardware; otherwise console lockups will occur.

9.1.5 Count for Cursor Blink

This parameter is the delay time between cursor blink. The normal value for proper TI-99 operation is 200. A good choice for use with a MYARC GENEVE at speed 5 is 500. The default is 200.

Enter a number between 1 and 999. The higher the number the slower the blink rate.

9.1.6 Memory Model

The Memory Model parameter describes how the editor, the compiler, and the debugger use the memory locations between >6000 and >7fff. Thereare three choices:

E/A : Editor Assember Cartridge
MINI-MEM : Mini-Memory Cartridge
SUPER-CA : Super-Cartridge (or MYARC GENEVE)

The E/A selection disables access of locations >6000 to >7fff. The MINI-MEM selection allows

memory between >6000 and >7fff.

If you are running with a memory dameon (e.g. BATCH-IT), you may want to disable via this selection the memory you are using for the dameon.

9.1.7 Background/Foreground Colors

This selection allows you to define the background (screen) color and the foreground (character) color. The default is white (foreground) on a blue background. When this item is selected, the following is displayed on the bottom of the screen:

```
Press 1 to Toggle Foreground
      2 to Toggle Background
```

Press the number "1" key to change to the next background color. Depress the number "2" key to change to the next foreground color. The screen and character colors, as well as the text describing the color, will change in response to pressing item 1 and 2. When you are satisfied with the color combination, press ENTER to exit.

9.1.8 Character for Cursor

This parameter defines the hexadecimal ASCII code for the cursor character. The default is z'5f', or the underscore.

9.1.9 Default Label for Printer

When your FORTRAN program is started, the PRINTER device as specified by the PRINTER name is opened for write access. It is bound to this FORTRAN unit number. The default number for this unit is 9.

9.1.10 Terminating Printer Character

If you use the wild card (asterick - *) for FORTRAN READ AND WRITE statements, this parameter defines the actual FORTRAN device number to be used. It defaults to 6 (the CRT device), but can be changed to match the Printer Unit Number (9) or any other unit number.

9.1.11 BOOT Disk Name

The BOOT Disk name defines the name which is prepended to the file name when an item is selected from the FORTRAN Main MENU, or when a file name is entered without a specific drive designation. The default is:

DSK.FORTCOMP.

If you keep the BOOT disk in one particular disk drive all of the time, or perhaps use a RAM disk, it would be worthwhile changing the name to that of the disk drive, e.g.:

DSK5.

If you have a MYARC HFDC Hard Disk, you will probably want to create a directory on the hard disk for all of the FORTRAN disks. For example, if you create a directory of HDS1.FORTCOMP., and copy all of the FORTRAN files there, you could enter the name of:

The OPEN routine (e.g. CALL OPEN) in 99 FORTRAN also prepends this file name to any file which is not given a specific drive designation. For example, if you place your libraries ML, FL, and GL in the FORTCOMP directory, you could reference them in the linker just by typing the names:

```
ML
FL
GL
```

instead of typing the full path names:

```
HDS1.FORTCOMP.ML
HDS1.FORTCOMP.FL
HDS1.FORTCOMP.GL
```

9.1.12 Printer

When your FORTRAN Program is started, a PRINTER device is opened for WRITE access. The name of the printer device opened is specified here. The default is blank (no printer). Some typical printer names which might go here are:

```
PIO
RS232/1.BA=4800.LF
```

9.1.13 Using Preferences

At the end of the preferences selection, you will be asked:

Shall I use Preferences (Y/N)?

To use the selected preferences, press *Y*. To ignore the preferences, press *N*.

Note that the preference selection will only be kept until you reboot, unless you save the menu using item 4, *Create New Menu.*

9.2 Disk Directory

The disk directory item provides a list of a selected disk drive or hard disk directory. When item 2 is selected from the utilities menu, the prompt:

```
Disk Directory
Enter Device Name (e.g. DSK1.)
```

Enter the name of the drive you want to catalog, or a hard disk directory name. The following are examples of several disk directory names:

```
DSK1.
DSK5.
HDS1.FORTCOMP.
```

The disk directory displays the current drive device name, the formatted disk name, the number of sectors available and used. It displays for each file the file name, the size in sectors, the type of file, and whether the file is protected or not.

9.3 Transform File

display/variable/80 or display/fixed/80.

ASCII and BINARY files transferred from other machines do not contain a "TIFILES" header which tells *FASTERM* and *TELCO* what type of file it is. *FASTERM* and *TELCO* create a file in DISPLAY/FIXED/128 format.

Transform takes the display/fixed/128 file and "unpacks" the file, and places it in the proper format. For display/variable/80, it also corrects *<CR>* and *<LF>* discrepancies, and detabs the file.

To use Transform, select item 3 on the utilities menu. The prompt:

99 Transform Utility

Enter Input File Name:

Enter the name of the file which is in display/fixed/128 format. The prompt:

Enter Output File Name:

will be displayed. Enter the name of the file which is to be created in display/variable/80 format or display/fixed/80 format. The prompt:

Display/Variable/80 File (Y/N)?

Enter a Y if this file is to be created as a display/variable/80 file. Enter a N if this file is to be created as a display/fixed/80 file.

The input file will be read and an output file created.

9.4 Creating New Menu

Item 4 on the Utilities Menu allows you to:

a. Create a new "UTIL1" file which will contain your saved preferences; so when you reboot 99 FORTRAN your changes will be saved.

b. Create a standalone BOOT image which can be used in conjunction with a linked 99 FORTRAN program to create an E/A 5 loadable program.

9.4.1 Creating a New BOOT Image

Item 1 on the MENU Image menu is to create a new UTIL1 boot image file. This will replace the existing UTIL1 file and the new file will contain your selected preferences.

When item 1 is selected, you will see the message:

Enter File Name to Save:
DSK.FORTCOMP.UTIL1

The file name "DSK.FORTCOMP.UTIL1" is dependent on what path name you specified in the Preferences selection. If this is where you want the UTIL1 file created, then press ENTER. If you want to place the UTIL1 file on another device, enter the full name, blank out any characters, and press ENTER.

For example, if you wanted to create a UTIL1 file on DSK1, you would enter:

DSK1.UTIL1 (and 8 spaces to blank out MP.UTIL1)

and press the ENTER button.

9.4.2 Create Standalone Image

Item 2 is the most confusing item in the utilities. Creating a standalone image requires some knowledge of how the Editor/Assembler creates item 5 "RUN PROGRAM FILE" programs.

When a 99 FORTRAN program is linked, it creates at least two executable files. These files are created in normal E/A 5 format; but they are missing the support routines necessary to run as a standalone program. When the program is run from the 99 FORTRAN MENU page, the support routines necessary are loaded in automatically when you select item 4 (Run) or item 5 (Run/Debug).

The "Create Standalone Image" creates a E/A 5 module which contains all of the support routines necessary to support a FORTRAN program; complete with your desired preferences.

To use the "Create Standalone Image" item, first link your FORTRAN program to create a normal TI-99 executable file. Then create the "Standalone Image" with the same name as the executable FORTRAN program, but with the last letter of the file name one letter less than the FORTRAN program name.

An example is best here. Suppose I want to create a standalone FORTRAN program called "KERMIT". I would link my 99 FORTRAN program and create executable modules as follows:

DSK1.KERMIU
DSK1.KERMIV

I would then enter the "Utilities", and select item "4" Create New Menu, select item "2" Create Standalone Image, and enter a file name of:

DSK1.KERMIT

A standalone image file would be created on DSK1 called KERMIT.

Now to execute the program, call up the Editor/Assembler menu, select item 5 "Run Program File", and enter the name of:

DSK1.KERMIT

Three files will be loaded (DSK1.KERMIT, DSK1.KERMIU, DSK1.KERMIV), and your FORTRAN program will be executed. When your program executes a STOP or CALL EXIT; then 99 FORTRAN will return your program to the TI Title Screen.

pp 0A-8, Section A.5 Screen Organizations (TI-99 GPL Only)

Add 80 column mode to the table as follows:

	32 column (graphics)	40 column (text)	80 column (text)	Bit Mode (graphic 2)
Number of Rows:	24	24	24	24
Number of Columns:	32	40	80	32
Character Size:	8x8	6x8	6x8	8x8
Sprites Allowed?	yes	no	no	yes
Screen Image Table Address:	0-2ff	0-3a1	0-77f	1800
Sprite Attribute List:	300-37f	n.a.	n.a.	1b00

blitnbbblit Color Table:	380-400	n.a.	n.a.	
Sprite Descriptor Table:	400-77f	n.a .	n.a.	
Sprite Motion Table:	780-7ff	n.a.	n.a.	
Character Pattern Table:	800-fff	800-fff	800-fff	0000

pp 0A-10, Section A.6.1 Subprogram Structure

Add the following to the end of section A.6.1:

If the function type is COMPLEX *8, then r5 and r6 must contain the real portion of the complex number, and r7 and r8 must contain the imaginary portion, i.e.:

```
MOV @REAL1,R5
MOV @REAL2,R6
MOV @IMAG1,R7
MOV @IMAG2,R8
```

If the function type is COMPLEX *16, then r5,r6,r7,r8 must contain the real portion of the complex number, and the eight byte area def'd as FACIM$ must contain the imaginary portion. For example:

```
REF  FACIM$

MOV @REAL1,R5
MOV @REAL2,R6
MOV @REAL3,R7
MOV @REAL4,R8
*
LI  R4,FACIM$
MOV @IMAG1,*R4+
MOV @IMAG2,*R4+
MOV @IMAG3,*R4+
MOV @IMAG4,*R4
```

pp 0A-11, Section A.6.2 Utilities

Remove the following entries from the address table:

DSKSS	20A4	Address of "Disk Type Devices" Table

Add the following entries to the utilities table:

GPLLNK	2084	Graphics Programming Language Interface
VRFR	205C	Vdp Read Registers

Add the following entries to the useful address table:

MEMTYP	20A8	Memory Type (0=E/A, 1=MINIMEM, 2=SUPER-CA)
MENVDP	202C	Last Location used in VDP RAM by MENU
MENU	2060	Menu Restart Entrance
PRTDOP	20A0	Address of Default Printer Name Start
PROGST	A000	Start Address of a 99 FORTRAN program
LOGSTR	A004	Logic Start Address (always >a000)
LOGEND	A006	Logic End Address
DATSTR	A008	Data Start Address
DATEND	A00A	Data End Address (COMMON Start)

Remove restrictions 1 (GPL environment) and 3 (uncompressed object). 99 FORTRAN now supports a GPL interface (GPLLNK) and also supports assembly language compressed object.

Add the following Appendices:

Appendix A.6 - Using PREDITOR with 99 FORTRAN

It is possible to use the *PREDITOR* text editor from Asgard Software in conjunction with the 99 FORTRAN package. To call *PREDITOR* directly from the FORTRAN MAIN MENU screen, you must make a few simple "patches" to *PREDITOR* using a sector editor. Neither the sector editor or *PREDITOR* is provided with 99 FORTRAN; you can purchase *PREDITOR* from Asgard Software, and there are many good sector editors available (e.g. part of the Advanced Diagnostics kit from Miller Graphics).

There are four cells within the *PREDITOR* program which must be patched, as follows (all values are in hexadecimal):

1. Removing >2000 to >2fff from *PREDITOR* workspace: This patch shrinks the memory block from >2000 to >3fff to >3000 to >3fff; so that the FORTRAN menu routines will not be written over by *PREDITOR*.

   The locations to patch are in the first sector:

Location	Was	Is
>0C	>20	>30
>10	>20	>30
>14	>1F	>0F

   This reduces the memory block starting at >3000 to >ffc bytes.

2. The next location to patch forces a return to the FORTRAN MENU routine from *PREDITOR*. Currently, *PREDITOR* using a GPL return mechinism through location >0070 in memory. This is changed to branch to location >2060; which is the MENU restart entry point. The sector to change is sector 3:

Location	Was	Is
>85	>00	>20
>86	>70	>60

   (On this patch, look for a >0460, >0070. This is a B @>0070 instruction, the return to the GPL interpreter. Patch this as indicated above, and it will generate a B @>2060 instruction).

After PREDITOR has been patched, copy the two *PREDITOR* modules PR and PS to the following names on your BOOT disk:

PR becomes FORT48A
PS becomes FORT48B

You can then select *PREDITOR* directly from the FORTRAN menu by selecting item 8 - USER Routine.

TI-99/4A GPL Mode

99 FORTRAN provides an interface to the 9918A graphic mode 2. Using graphic mode 2, the screen is organized as 256 pixels wide by 192 pixels deep. Using the SETPIX subroutine, you can turn on any pixel on the screen. Using the SETVEC subroutine, you can draw lines on the screen given any two sets of x,y coordinates. Using the CIRCLE subroutine, you can draw circles on the screen.

Note that squeezing bitmap mode into the 99 FORTRAN environment is a little difficult, due to the shortage of VDP memory. To make room for the bitmap mode, you must preceed the setmod statment which sets up the bitmap mode with a call files(1) statement (meaning, of course, you can only open a single floppy disk file from within your program).

For example:

```
call files( 1 )          ! free up vdp memory
call setmod( 4 )         ! set graphics mode 2
call screen( 7, 2 )      ! set color black on red
```

If you do not use the call files(1) statement, you will get a FORTRAN execution error. While each pixel on the screen is individually addressable, the color of each pixel is addressable in groups of eight horizontal pixels. For example:

```
                 col
                 0123456789.....          255
row 0            ....*.....               .....
row 1            ..........               .....
  .
  .
row 191          ..........               .....
```

Turning on the pixel at x=0, y=4 (row 0, column 4) using the statement:

```
call setpix ( 0, 4, 7 )
```

will cause the pixel noted with the asterick to be turned ON using color 7 (dark red) and if any of the other pixels 0,0 through 0,7 are ON, will also change their color to dark red.

It is possible to skip setting the color of the pixel, i.e.:

```
call setpix ( 0, 4 )
```

will turn on the pixel at 0,4, without changing the currently defined colors for the group of eight pixels.

Other subroutines which are operative in the bitmap mode are call clear and call screen.

Call clear clears the screen by turning OFF all of the pixels on the screen. The call screen subroutine sets all of the colors in the color table to the requested color settings. If the foreground color is not specified, the foreground color is set using the default foreground color from the "preferences" utility.

A unique feature of the 99 FORTRAN setmod(4) is the text capability. You can write text to the screen via normal FORTRAN write statements, i.e.:

```
      call files(1)          ! free up vdp memory
      call setmod(4)         ! set graphics mode 2
      write ( 6, 9100 )
9100 format ( '+', m10.10, 'this is text' )
```

will write the text string 'this is text' at row 10, column 10, as if you were in graphics mode 1. The screen is organized for text mode as 32 columns wide by 24 rows deep. The following restrictions apply to text mode in graphics mode 2:

a. There is no scrolling, when the end of the page is reached, the invisible cursor is positioned to the top of the page.

b. You cannot redefine the character set via CALL CHAR. You can redefine the ASCII character set before calling the SETMOD routine, i.e.:

```
call char ( Z'0030', Z'F0F0F0F0F0F0F0F0' )
call setmod ( 4 )
```

would redefine the shape of the ASCII character '0'.

c. You can only use the ASCII characters. If you write a character which is not defined in the normal ASCII character set (from z'0020' to z'007e'), it will be interpreted as a blank (space).

Several other restrictions of the bitmap mode are:

a. You cannot use the symbolic debugger (there just isn't enough RAM to run it!)

b. You cannot use sprites

MDOS Mode

Within MDOS, seven graphic modes are supported (graphics modes 1 through 7), two text modes, and the multi-color mode. Since 9640 FORTRAN uses the MDOS Video XOPs whenever possible, the concepts of paging are supported.

It is recommended to the serious user that the 9938 MSX-Video Data Processor User Manual be purchased. This manual is currently the only source of information concerning the 9938 chip that describes all 9938 video modes.

Since the 9640 FORTRAN Debugger uses its own video mode (80 column video) outside of MDOS, any screen mode can be used within your FORTRAN program with the debugger.

** * * end of errata pages * * **