

# **Addendum 2.0:8**

**to**

## **fbForth 2.0**

**A File-Based Cartridge Implementation  
of TI Forth**

**Lee Stewart**

**June 20, 2016**

Copyright © 2014 – 2016 by Lee Stewart

Permission is hereby granted to copy and distribute this document at will.

# Table of Contents

1	Introduction.....	1
2	Startup Changes.....	2
2.1	The Opening Menu.....	2
2.2	Enabling 1024-Byte SAMS Mapping.....	2
2.3	Changes to the fbForth 2.0 ISR.....	2
2.4	Changes to COLD.....	3
2.5	Redefinition of BOOT.....	3
3	Interrupt Service Routines (ISRs).....	4
3.1	Overview of fbForth 2.0:8's ISR.....	4
3.2	A Detailed Look at fbForth 2.0:8's ISR.....	4
3.3	Installing a User ISR.....	5
3.4	Example of a User ISR: DEMO.....	6
3.4.1	Installing the DEMO ISR.....	7
3.4.2	Un-Installing the DEMO ISR.....	7
3.5	Some Additional Thoughts Concerning the Use of ISRs.....	8
4	Screen Font Changes.....	9
5	TI Forth Block Utilities.....	10
5.1	TIFBLK: Display TI Forth Block.....	10
5.2	TIFIDX: Display TI Forth Index Lines.....	11
5.3	TIF2FBF: Copy TI Forth Blocks to fbForth Blocks.....	12
5.4	TIFVU: TI Forth Browser/Copier.....	12
6	Bug Fixes.....	15
Appendix A	The fbForth 2.0 Glossary.....	16
A.1	fbForth 2.0 Word Descriptions.....	16
Appendix B	User Variables in fbForth 2.0.....	29
B.1	fbForth 2.0 User Variables (Address Offset Order).....	29
B.2	fbForth 2.0 User Variables (Variable Name Order).....	31
Appendix C	fbForth 2.0 Load Option Directory.....	33
C.1	Option: 64-Column Editor.....	33
C.2	Option: CPYBLK -- Block Copying Utility.....	33
C.3	Option: Memory Dump Utility.....	33
C.4	Option: TRACE -- Colon Definition Tracing.....	33
C.5	Option: Printing Routines.....	33
C.6	Option: TMS9900 Assembler.....	34
C.7	Option: CRU Words.....	34
C.8	Option: More Useful Stack Words etc.....	34
C.9	Option: Stack-based String Library.....	34
C.10	Option: DIR -- Disk Catalog Utility.....	34
C.11	Option: CAT -- Disk Catalog Utility.....	34
C.12	Option: TI Forth Block Utilities.....	34
Appendix D	Contents of FBLOCKS.....	35



# 1 Introduction

This addendum has “2.0:8” in its title because it was published after the release of **fbForth 2.0:8** and is intended as an interim update to **fbForth 2.0: A File-Based Cartridge Implementation of TI Forth**, which will be updated in due course.

Detailed in this addendum are bug fixes, words no longer part of **fbForth 2.0**, modified words and new words. Some chapter(s) and appendices of the manual have been rewritten *in toto* and reference the relevant chapter or appendix of the edition of **fbForth 2.0: A File-Based Cartridge Implementation of TI Forth**.

Though I have been careful in coding **fbForth 2.0**, as with anything else in this document, you assume responsibility for any use you make of it. Please, feel free to contact me with comments and corrections at [lee@stewkitt.com](mailto:lee@stewkitt.com).

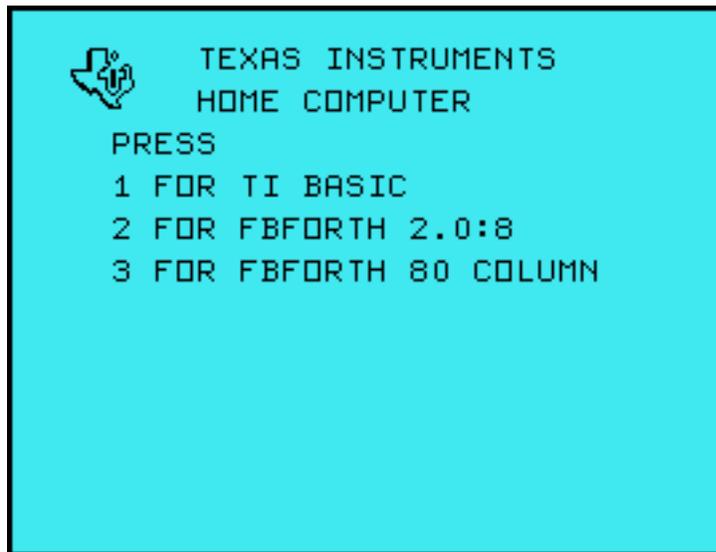
—Lee Stewart  
June, 2016  
Silver Run, MD

## 2 Startup Changes

This chapter will detail the startup changes for **fbForth 2.0** since the manual was last updated.

### 2.1 The Opening Menu

The opening menu has two choices for **fbForth 2.0** shown in the screen shot below:



- Option 2 will open in 40-column Text mode, **TEXT** .
- Option 3 will open in 80-column Text mode, **TEXT80** , which must not be selected unless the user has an F18A, V9938 or similar video display processor capable of 80-column Text mode. Otherwise, the display will be corrupted and VRAM will be improperly set up.

### 2.2 Enabling 1024-Byte SAMS Mapping

After selection of one of the **fbForth 2.0** options, the first thing that the initialization code does is to set up 1024-byte SAMS, whether or not a SAMS card is present! Then, it tests for proper SAMS operation by writing a 16-bit value to an arbitrary address, mapping another SAMS bank to the 4KiB segment containing the written value and, finally, testing for the written value. If the written value is found, the SAMS mapping did not work. If it is not found, SAMS mapping obviously worked. The SAMS flag is set to 0 or 1, accordingly. It is this flag that is tested by **SAMS?** , see Appendix A “The fbForth 2.0 Glossary”.

### 2.3 Changes to the **fbForth 2.0** ISR

The **fbForth 2.0** ISR is now enabled at startup so that the new speech ( **SAY** and **STREAM** ) and sound word ( **PLAY** ) will work. The speech and sound word ISRs are driven by the **fbForth 2.0**

ISR. It is easy enough to disable it if the user does not use speech, sound or a user ISR and wants to recover the little bit of time it takes for the **fbForth 2.0** ISR to check for non-existent ISRs to service. See Chapter 3 “Interrupt Service Routines (ISRs)” for details.

## 2.4 Changes to COLD

**COLD** is the last routine executed by the **fbForth 2.0** startup code. Formerly, it was a high-level Forth word that called another high-level Forth word ( **BOOT** ) at its conclusion. They have both been combined into a single ALC routine that (re-)sets the Forth environment to the default startup conditions.

Holding down a key immediately upon execution of **COLD** will force **COLD** to look for FBLOCKS from that disk. If the user executed **COLD**, the last loaded font is reloaded regardless of the new disk selection indicated by the held key; whereas, at startup, the held key depressed immediately after the menu selection, will, in fact, also cause the search for FBFONT on the held key’s disk. Both invocations of **COLD** will not attempt to load FBLOCKS if **<ENTER>** is the held key.

If **<ENTER>** was held down at powerup or after execution of **BOOT** (see next section), the default disk drive for both FBLOCKS and FBFONT is DSK1. Though DSK1.FBLOCKS is not loaded, DSK1.FBFONT *is* loaded (or, at least, attempted). For the next go-round with **COLD** executed by the user, if no key is held down then, both DSK1.FBLOCKS and DSK1.FBFONT will be loaded if found.

User changes to the following settings will survive a user-executed **COLD** :

- Display font (see **USEFFL**, **SCRFONT**, **FNT** for how to change font)
- Default colors for all VDP modes (see definition of **DCT** for how to change)
- Default VDP mode, which should be limited to **TEXT80** (0) or **TEXT** (1) (see definition of **DCT** for how to change)
- Default **S0** and **TIB** changed by **S0&TIB!** .

## 2.5 Redefinition of BOOT

**BOOT** has been redefined to restart **fbForth 2.0** at the cartridge startup code. The desired default VDP text mode of **TEXT80** or **TEXT** may be set by pushing to the stack 0 or 1, respectively, prior to executing **BOOT** :

**0 BOOT**

will set the default VDP text mode to **TEXT80** and reboot **fbForth 2.0** just as though the user had made the selection on the opening screen.

**BOOT** may also be executed with nothing on the stack:

**BOOT**

which will set the default VDP text mode to **TEXT** as though the user had executed

**1 BOOT**

Holding a disk-selection key or **<ENTER>** will have the same effect as at powerup selection.

## 3 Interrupt Service Routines (ISRs)

This chapter is a complete replacement of the corresponding Chapter 10 of the manual.

### 3.1 Overview of *fbForth 2.0:8's* ISR

Though the user may disable it<sup>1</sup>, **fbForth 2.0's** ISR is now hooked at startup and is executed for every interrupt. There are three entry points into **fbForth 2.0's** ISR. Their ALC labels are **INT1**, **INT2** and **INT3**.

**INT1** is where the console ISR branches at the end of its interrupt processing. It processes any pending speech (started with **SAY** or **STREAM**) and sound (started with **PLAY**). It then looks to see whether a user ISR is installed in user variable **ISR**. If so, it modifies the **fbForth 2.0** inner interpreter's **NEXT** (R15) to re-enter at **INT2**.

Re-entry at **INT2** will restore **NEXT** and set up re-entry yet again at **INT3** for cleanup before exiting by branching to the user ISR.

When the user ISR finishes, **fbForth 2.0's** ISR is re-entered at **INT3** for cleanup via the inner interpreter. Upon exit, the inner interpreter will resume processing Forth words where it was interrupted.

A user ISR will be executed only if the user has installed an ISR using the steps detailed in § 3.3 "Installing a User ISR".

### 3.2 A Detailed Look at *fbForth 2.0:8's* ISR

The console ISR branches to the contents of **83C4h** (R2 of the console ISR workspace [**83C0h**]) if it is non-zero. In **fbForth 2.0:8**, **83C4h** contains the address of ISR entry point **INT1** (currently, **3020h**) mentioned in the last section. This same entry point is in user variable **INTLNK**, as well. This means that the console ISR will branch to the **fbForth 2.0** ISR with **BL \*R12** through the GPL workspace (**83E0h**), R12 containing the ISR's entry point.

Upon entry at **INT1** from the console ISR, the **fbForth 2.0** ISR does the following:

- Checks for pending speech and sound. If found, the following ISR branch stack is set up and executed:
  - Relevant speech ISR address, if speech pending;
  - Sound-list #1 ISR address, if pending;
  - Sound-list #2 ISR address, if pending;
  - **fbForth 2.0** ISR return address.
- Restores interrupted bank.
- Checks user variable **ISR** for a non-zero value, implying a user ISR is installed. If a user ISR is defined, modifies **NEXT** to re-enter **fbForth 2.0's** ISR at **INT2** at the next branch through **NEXT** via **B \*NEXT** or **B \*R15**, which will set up to execute the user ISR.

---

<sup>1</sup> **fbForth 2.0's** ISR may be disabled by zeroing **83C4h** with **HEX 0 83C4 !**.

- Exits the **fbForth 2.0** ISR by changing to the ISR workspace (**83C0h**) and returning to the caller of the console ISR.

Upon entry at **INT2** (because we have a user ISR defined), the **fbForth 2.0** ISR does the following:

- Disables interrupts via **LIMI 0**.
- Disables VDP interrupt.
- Restores **NEXT** to its value before it was changed at **INT1**.
- Sets the **fbForth** “pending interrupt” flag.
- Pushes current IP (next word pointer) to the return stack.
- Changes IP to **INT3** for cleanup re-entry to **fbForth 2.0** ISR.
- Copies the value in **ISR** to W (current word pointer) so inner interpreter will execute the user ISR.
- Branches to inner interpreter to execute user ISR via **DOEXEC**.

Upon entry at **INT3** (because we are returning from executing the user ISR), the **fbForth 2.0** ISR does the following:

- The inner interpreter actually branches to the address 4 bytes after **INT3**, which pops the saved IP from the return stack.
- Clears the **fbForth** “pending interrupt” flag.
- Clears the pending VDP interrupt by reading VDP status.
- Re-enables VDP interrupt.
- Re-enables interrupts via **LIMI 2**.
- Branches to inner interpreter via **NEXT** to continue executing the interrupted list of word addresses.

If the user’s ISR (see below) is properly installed, **fbForth 2.0**’s ISR, at interrupt, modifies **NEXT** so that the very next time **B \*NEXT** or **B \*R15** is executed from **fbForth 2.0**’s workspace, **fbForth 2.0**’s ISR is re-entered to disable interrupts and to insert execution of the user ISR and its cleanup into the **fbForth 2.0** inner interpreter’s list of execution addresses (CFAs).

The TI-99/4A has the built-in ability to execute an interrupt routine every 1/60 second. This facility has been extended by the **fbForth 2.0** system so that the routine to be executed at each interrupt period may be written in Forth rather than in assembly language. This is an advanced programming concept and its use depends on the user’s knowledge of the TI-99/4A.

### ***3.3 Installing a User ISR***

The user variables **ISR** and **INTLNK** are provided to assist the user in using ISRs. Initially, **INTLNK** contains the address of the address of the **fbForth 2.0** ISR handler and **ISR** is set to 0 to indicate no user ISR. To correctly use user variable **ISR**, the following steps should be followed:

Step	Forth Code
1) Create and test an <b>fbForth 2.0</b> routine to perform the function. Let's call it <b>MYISR</b> :	<b>: MYISR ... ;</b>
2) Clear the <b>fbForth 2.0</b> ISR hook to temporarily disable it:	<b>HEX 0 83C4 !</b>
3) Determine the Code Field Address (CFA) of the routine in (1):	<b>' MYISR CFA</b>
4) Write the CFA from (3) (still on the stack) into user variable:	<b>ISR : ISR !</b>
5) Write the contents of <b>INTLNK</b> into <b>83C4h (33732)</b> to re-enable the <b>fbForth 2.0</b> ISR:	<b>HEX INTLNK @ 83C4 !</b>

The ISR linkage mechanism is designed so that your interrupt service routine will be allowed to execute immediately after each time the **fbForth 2.0** system executes the instruction whose address is in **NEXT** (as it does at the end of each code word). In addition, the **KEY** routine has been coded so that it also executes through **NEXT** after every keyscan whether or not a key has been pressed. The execution of the "NEXT" instruction in the inner interpreter is actually coded in TI Assembler as **B \*NEXT** or **B \*R15** because **fbForth 2.0** workspace register 15 (**R15** or **NEXT**) always contains the address of "NEXT" (**MOV \*IP+,W**) except, of course, when we temporarily force its change by installing a user ISR. This executes the same procedure as the **fbForth 2.0** Assembler words **;ASM** and **NEXT**, (see Chapter 9 of the manual).

Before installing an ISR, you should have some idea of how long it takes to execute, keeping in mind that for normal behavior it should execute in less than 16 milliseconds. ISRs that take longer than that may cause erratic sprite motion, speech and sound because of missed interrupts. In addition it is possible to bring the **fbForth 2.0** system to a slow crawl by using about 99% of the processor's time for the ISR.

The ISR capability has obvious applications in game software as well as for playing background music or for spooling blocks from file to printer while other activities are taking place. This final application will require that file buffers and user variables for the spool task be separate from the main Forth task or a very undesirable cross-fertilization of buffers may result. In addition it should be mentioned that disk activity causes all interrupt service activity to halt.

ISRs in **fbForth 2.0** can be written as either colon definitions or as **ASM:** definitions. The former permits very easy routine creation, and the latter permits the same speed capabilities as routines created by the Editor/Assembler. Both types can be used in a single routine to gain the advantages of both.

### 3.4 Example of a User ISR: DEMO

An example of a simple ISR is given below. This example also illustrates some of the problems associated with ISRs and how they can be circumvented. The problems are:

- 1) A contention for PAD between a normal Forth command and the ISR routine.
- 2) Long execution time for the ISR routine. (Even simple routines, especially if they include output conversion routines or other words that nest Forth routines very deeply, will not complete execution in 1/60 second.)

The problem listed in (1) is overcome by moving PAD in the interrupt routine to eliminate the interference between the foreground and the background task. An example of problem (2) would be attempting to use the built-in number formatting routines, which are quite general and, hence, pay a performance penalty. **DEMO** performs this conversion rather crudely, but fast enough that there is adequate time remaining in each 1/60 second to do meaningful computing.

```

0 VARIABLE TIMER           (TIMER will hold the current count)
: UP 100 ALLOT ;           (move HERE and thus PAD up 100 bytes)
: DOWN -100 ALLOT DROP ;   (restore PAD to its original location)
: DEMO UP                  (move PAD to avoid conflict)
  1 TIMER +! TIMER @        (increment TIMER , leave on stack)
  PAD DUP 5 +               (ready to loop from PAD + 5 down to PAD + 1)
  DO
    0 10 U/                 (make positive double, get 1st digit)
    SWAP 48 +              (generate ASCII digit)
    I C!                   (store to PAD )
    -1 +LOOP               (decrement loop counter)
    PAD 1+ SCRN_START @ 5 VMBW (write to screen)
    DOWN ;                 (restore PAD location)

```

### 3.4.1 Installing the DEMO ISR

To install this ISR, the following code should be executed:

```

HEX 0 83C4 !             (clear console ISR hook)
' DEMO CFA              (get CFA of the word to be installed as user ISR)
ISR !                   (place it in user variable ISR )
INTLNK @                (get the fbForth 2.0 ISR address to the stack)
HEX 83C4 !             (re-install fbForth 2.0 ISR into console ISR hook)
                          (Note: the CFA of DEMO must be in user variable ISR
                          before writing to 83C4h)

```

### 3.4.2 Un-Installing the DEMO ISR

To reverse the installation of the ISR, the following code should be executed:

```

HEX 0 83C4 !             (clear console ISR hook)
0 ISR !                 (disable user ISR by zeroing user variable ISR )
INTLNK @                (get the fbForth 2.0 ISR address to the stack)
HEX 83C4 !             (re-install fbForth 2.0 ISR into console ISR hook)

```

### ***3.5 Some Additional Thoughts Concerning the Use of ISRs***

ISRs are uninterruptible. Interrupts are disabled by the code that branches to your ISR routine and they are not enabled until just before branching back to the foreground routine. *Do not enable interrupts in your interrupt routine.*

- 1) Caution must be exercised when using PABs, changing user variables or using disk buffers in an ISR, as these activities will likely interfere with the foreground task unless duplicate copies are used in the two processes.
- 2) An ISR must never expect nor leave anything on the stacks. It may however use them in the normal manner during execution.
- 3) Disk activity disables interrupts as do most of the other DSRs in the TI-99/4A. An ISR that is installed will not execute during the time interval in which disk data transfer is active. It will resume after the disk is finished. Note that it is possible to **LOAD** from disk while the ISR is active. It will wait for about a second each time the disk is accessed. The dictionary will grow with the resultant movement of **PAD** without difficulty.

## 4 Screen Font Changes

The default Screen Font File with descenders for ASCII character values 0 – 127 (1024 bytes) is no longer in ROM. Now, it is to be found in DSK1.FBFONT unless the default disk has been changed at bootup. The boot DSK #, *n*, is saved as the 5<sup>th</sup> byte of the packed string for the default blocks filename, “DSK*n*.FBLOCKS”, the 1<sup>st</sup> byte of which is the string length.

At powerup,

- **SCRFONT** is set to its new default value of -1;
- The default font is loaded from DSK*n*.FBFONT by **FNT** .

At non-powerup **COLD** , the font file loaded before **COLD** was invoked is reloaded, unless the user changed the default value of **SCRFONT** to 0. The default value of **SCRFONT** can be changed to 0 to force loading the console font, with its small caps for lowercase, with the following code:

```
0 UCONS$ @ 68 + !
```

**UCONS\$** is the address of the default-value table of User Variables and 68 (44h) is **SCRFONT** 's position in the table.

If the default font file cannot be found,

- **SCRFONT** and its default value are set to 0;
- The console font, with small caps for lowercase, is loaded.

If **SCRFONT** ≠ 0, **FNT** loads the default font file, the PAB for which follows the **fbForth 2.0** disk buffer, **DISK\_BUF** , in VRAM.

The user can change the default font to come from a binary font file of the user's choosing with **USEFFL** . **USEFFL** will set up the font-file PAB (immediately follows **DISK\_BUF** in VRAM). The default font filename will be copied to the font PAB in VRAM.

The **fbForth 2.0** word **FNT** loads either the default font file (can be changed by user) or the console font into the Pattern Descriptor Table (PDT) depending on the value of the user variable **SCRFONT** . The default font is loaded from DSK1.FBFONT by **FNT** (or from DSK*n*.FBFONT if key *n* is held down) at **fbForth 2.0** startup because **SCRFONT** = -1 at startup. The **fbForth 2.0** system default font contains the patterns for ASCII character codes 0 – 127. The font pattern for each character is 8 bytes, which means that 1KiB of pattern code is loaded into the PDT. This font contains true lowercase characters with true descenders.

It should be noted that each time the VDP mode is changed (except for Graphics2 [bitmap]), the current screen font is reloaded. The user can always change the value of **SCRFONT** to 0 to force (re)loading the console screen font. Changing **SCRFONT** back to a non-zero value will switch font loading to the currently stored font-file name, be it the system or user font file.

## 5 TI Forth Block Utilities

Words introduced in this chapter:

**TIF2FBF**    **TIFBLK**    **TIFIDX**    **TIFVU**

The TI Forth Block Utilities are not part of the resident dictionary so must be loaded from FBLOCKS (see current FBLOCKS file **MENU** for TI Forth Block Utilities). They are provided to make it easy to view TI Forth blocks (called “screens” in TI Forth), index lines of a range of blocks and copy a range of blocks to an **fbForth 2.0** blocks file. The utilities listed in the first three sections below perform these functions individually. The last section presents a browser/copier that is patterned after the **fbForth 2.0** block editors.

Note that “IS:” is short for “Inout Stream”.

### 5.1 TIFBLK: Display TI Forth Block

**TIFBLK**    (IS:blk DSKn)

**TIFBLK** displays block *blk* from disk *DSKn*. The display may be paused/resumed by tapping any key except **<BREAK>**, which will abort the display. The display is automatically paused if the block cannot be displayed all at once.

The following shows the first screen of a Text mode example displayed with the Forth code just before the screen shot:

**TIFBLK 11 DSK2**

```

0! ( JDRAW continued...)
1! BASE->R HEX
2! : JUP_CUR JPEN @ IF 5 0 SPRCOL ELSE
F 0 SPRCOL ENDIF ;
3! : JER_CUR F890 A0C0 8000 0000 10 SPC
HAR ;
4! : JDR_CUR F8F0 E0C0 8000 0000 10 SPC
HAR ;
5! : JMENU FF 1 SPRPUT 1F F 2 SPRPUT F
1F 3 SPRPUT 1F 1F 4 SPRPUT
6! BEGIN KEY DUP CASE ( Toggle DMOD
E!JPEN!Quit & leave key)
7! 44 OF DMODE @ 1 XOR DUP DMO
DE !
8! IF JER_CUR ELSE JDR_CU
R ENDIF ENDOF
9! 50 OF JPEN @ 1 XOR JPEN ! J
UP_CUR ENDOF
10! 51 OF ( Just leave 'Q') EN
DOF
11! DROP 0 SWAP ENDCASE ( Leave 0
if illegal key)

```

The display was paused after twelve lines were displayed due to wrapping of 64-character lines on the 40-character display. Tapping a key will continue the display of the remaining four lines.

The following is the same example in Text80 mode using the same Forth code as above:

```

0! ( JDRAW continued...)
1 BASE->R HEX
2 : JUP_CUR JPEN @ IF 5 0 SPRCOL ELSE F 0 SPRCOL ENDIF ;
3 : JER_CUR F890 80C0 8000 0000 10 SPCHAR ;
4 : JDR_CUR F8F0 E0C0 8000 0000 10 SPCHAR ;
5 : JMENU F F 1 SPRPUT IF F 2 SPRPUT F 1F 3 SPRPUT IF 1F 4 SPRPUT
6   BEGIN KEY DUP CASE ( Toggle DMODE!JPEN!Quit & leave key)
7     44 OF DMODE @ 1 XOR DUP DMODE !
8     IF JER_CUR ELSE JDR_CUR ENDIF ENDOF
9     50 OF JPEN @ 1 XOR JPEN ! JUP_CUR ENDOF
10    51 OF ( Just leave 'Q') ENDOF
11    DROP 0 SWAP ENDCASE ( Leave 0 if illegal key)
12    -DUP UNTIL ( Leave value if legal key)
13    5 1 DO 10 0D0 I SPRPUT LOOP ( Hide menu) ; R->BASE -->
14
15
ok:0

```

## 5.2 TIFIDX: Display TI Forth Index Lines

**TIFIDX** (IS:*strtBlk endBlk DSKn*)

**TIFIDX** displays the index lines (first lines) of a range of TI Forth blocks (*strtBlk* to *endBlk*) from disk *DSKn*. The display may be paused/resumed by tapping any key except **<BREAK>**, which will abort the display. The display is automatically paused if the block cannot be displayed all at once.

The following shows the first screen of a Text mode example:

**TIFIDX 10 15 DSK2**

```

10! CR ." LOADING JDRAW---" CR ." Joy
stick drawing program..."
11! ( JDRAW continued...)

12! ( JDRAW continued...)

13! ( JDRAW continued...)

14!

15!

...done ok:0

```

The index line, (line #0) of each block from block #10 – #15 is listed above. Had more than 12 blocks (64 characters each index line) been selected, the display would have paused as for **TIFBLK** in the previous section.

The following is the same example in Text80 mode:

```

10: CR ." LOADing JDRAM--" CR ." Joystick drawing program..."
11: ( JDRAM continued...)
12: ( JDRAM continued...)
13: ( JDRAM continued...)
14:
15:
...done ok:0

```

### 5.3 TIF2FBF: Copy TI Forth Blocks to fbForth Blocks

**TIF2FBF** (IS:*srcStrtBlk srcEndBlk DSKn dstStrtBlk dstFile*)

**TIF2FBF** functions in much the same way as **CPYBLK**. The format of the command is the same except that the source is *DSKn*, not a filename. The *n* of *DSKn* is the disk number of the TI Forth disk. The destination *dstFile* must be the name of an existing blocks file (see **MKBFL** to create one). The following command will copy blocks 4 – 7 from TI Forth DSK2 to blocks 10 – 13 of DSK1.MYBLOCKS:

```
TIF2FBF 4 7 DSK2 10 DSK1.MYBLOCKS
```

### 5.4 TIFVU: TI Forth Browser/Copier

**TIFVU** (IS:*blk DSKn*)

Browse TI Forth blocks and, optionally, copy a range of blocks to an **fbForth** blocks file. The browser is interactive with the following functions:

Key	Function
<FCTN+4>	View next block.
<FCTN+6>	View previous block.
<FCTN+D>	View the next panel for Text mode—ignored in Text80 mode.
<FCTN+S>	View the previous panel for Text mode—ignored in Text80 mode.
<FCTN+T>	View a specific TI Forth block number.
<FCTN+F>	Specify a destination <b>fbForth</b> block number for next copy.
<CTRL+F>	Specify a destination <b>fbForth</b> blocks file, which must already exist.
<CTRL+S>	Copy a range of blocks starting from the displayed TI Forth block to the displayed destination <b>fbForth</b> block. You are prompted for the number of blocks to copy after selecting this command.
<FCTN+9>	Exit the browser.

Following is an example of the browser/copier in Text mode, which shows three panels of the same block:

```
TIFVU 12 DSK2
```

```

=====TI Forth Block Viewer/Copier=====
TI Forth:DSK3  fbForth:
Block 12      Block
0  ( JDRAW continued...)
1  BASE->R HEX
2  : J_DIR ( n1 --- n2 ) ( Change FC!
3  CASE FC OF -1 ENDOF
4  4 OF 1 ENDOF
5  0 SWAP ENDCASE ; ( ill
6  : JLIM ( n1 n2 --- n3 ) SWAP DUP 0
7  DUP ROT < IF SWAP DROP ELSE DR
8  : JDRAW J_INIT BEGIN 1 JOYST ROT
9  CASE
10 FF OF DROP DROP 0 ENDOF
11 12 OF DROP DROP JMENU 51 =
12 DROP J_DIR MINUS JDR @ + BF
13 JLIM JDC ! JDC @ JDR @ 0
14 JPEN @ IF JDC @ JDR @ DOT
15 UNTIL TEXT ;

F4:+Block F6:-Block FD:+Panel FS:-Panel
FT:TI# FF:fb# ^F:BlkFil ^S:TI>fb F9:xit
    
```

Left panel showing columns 0 – 33.

Middle panel showing columns 15 – 48.

```

=====TI Forth Block Viewer/Copier=====
TI Forth:DSK3  fbForth:
Block 12      Block
0 ed...)
1
2 - n2 ) ( Change FC!0:4 to -1:0:1)
3 -1 ENDOF
4 1 ENDOF
5 P ENDCASE ; ( illegal!down key)
6 --- n3 ) SWAP DUP 0< IF DROP DROP
7 F SWAP DROP ELSE DROP ENDIF ENDIF
8 BEGIN 1 JOYST ROT ( Main progr
9
10 OP DROP 0 ENDOF
11 OP DROP JMENU 51 = ENDOF
12 IR MINUS JDR @ + BF JLIM JDR ! J_D
13 DC ! JDC @ JDR @ 0 SPRPUT
14 IF JDC @ JDR @ DOT ENDIF 0 0 ENDC
15 ;

F4:+Block F6:-Block FD:+Panel FS:-Panel
FT:TI# FF:fb# ^F:BlkFil ^S:TI>fb F9:xit
    
```

Right panel showing columns 30 – 63.

```

=====TI Forth Block Viewer/Copier=====
TI Forth:DSK3  fbForth:
Block 12      Block
0
1
2 FC!0:4 to -1:0:1)
3
4
5 illegal!down key)
6 UP 0< IF DROP DROP 0 ELSE DUP ROT
7 E DROP ENDIF ENDIF ;
8 OT ( Main program loop)
9
10
11 1 = ENDOF
12 + BF JLIM JDR ! J_DIR JDC @ + FF
13 @ 0 SPRPUT
14 DOT ENDIF 0 0 ENDCASE
15 R->BASE -->

F4:+Block F6:-Block FD:+Panel FS:-Panel
FT:TI# FF:fb# ^F:BlkFil ^S:TI>fb F9:xit
    
```

And, here is the same example in Text80 mode:

```

=====TI Forth Block Viewer/Copier=====
TI Forth:ISK2 fbForth:
Block 12 Block
0 ( JDRAW continued...)
1 BASE->R HEX
2 : J_DIR ( n1 --- n2 ) ( Change FC:0:4 to -1:0:1)
3 CASE FC DF -1 ENDOF
4 DF 1 ENDOF
5 0 SWAP ENDCASE ; ( illegal!down key)
6 : JLIM ( n1 n2 --- n3 ) SWAP DUP 0< IF DROP DROP 0 ELSE DUP ROT
7 DUP ROT < IF SWAP DROP ELSE DROP ENDF ENDF ;
8 : JDRAW_J_INIT BEGIN 1 JDYST ROT ( Main program loop)
9 CASE
10 FF DF DROP DROP 0 ENDOF
11 12 DF DROP DROP JMENU 51 = ENDOF
12 DROP J_DIR MINUS JDR @ + BF JLIM JDR ! J_DIR JDC @ + FF
13 JLIM JDC ! JDC @ JDR @ 0 SPRPUT
14 JPEN @ IF JDC @ JDR @ 0 ENDF ENDF 0 0 ENDCASE
15 UNTIL TEXT ; R->BASE -->
F4:+Block F6:-Block FD:+Panel FS:-Panel FT:TI# FF:fb# ^F:Blkfil ^S:TI>fb F9:xlit

```

## 6 Bug Fixes

The following bug fixes have been made over a period of time. They are in no particular time order.

-  The insert-blank-line function, `<CTRL+8>`, in the 40/80-column editor would not blank the entire new line if the cursor were not located in the first column.
-  The character-copy function in the 40/80-column editor would cause **fbForth 2.0** to crash if the line-insertion and line-deletion functions were used on the last line of a block. The problem was not testing for a copy-count of 0 before copying the first character, causing the count to pass 0 before the test if the function was passed a count of 0, which it is on the last line.
-  **SGN** would yield +1 for -32768 (**8000h**), the largest single-precision (16-bit) negative number possible on the TI-99/4A.
-  **SSDT** was improperly setting the address of the Sprite Pattern Descriptor Table. **SSDT** is the easiest way for a user to change the Sprite Pattern Descriptor Table in graphics mode to a different location from the the default **800h**. The default, **800h**, is coincident with the text Pattern Descriptor Table. It is easy enough to change the **SSDT** in code, but it is not trivial. Besides, **SSDT** not only changes the user variable read by the constant, **SPDTAB**, but also changes VDP register #6 to the proper value and executes **DELALL** to initialize sprites.
-  **SPRPUT** was setting the x position to 255 (rightmost position) if y was 0.
-  **MOTION** was setting the x | y vector to -1 if the y | x vector was negative.
-  If sprite automotion was not stopped in Graphics mode, blinking text appeared in Text, Text80, Bitmap and Split modes. Automotion was not stopped when changing VDP modes. For some reason, if sprite automotion is enabled and sprites are left defined, Text80, Bitmap and Split modes show blinking areas on the screen that correspond to those sprites, particularly those defined with patterns in the text PDT area.
-  **BSAVE** was not explicitly saving the pointer to the last word in each of the Forth and Assembler vocabularies.
-  **BSAVE** and **BLOAD** were not saving and loading, respectively, the vocabulary link fields of the Forth and Assembler vocabularies.
-  **DELALL** was only marking the first 8 sprites as deleted, *i.e.*,  $y = \mathbf{D0h}$ , when it should have been doing it for all 32! The upshot of this bug was that, as soon as sprite #7 was defined, all of the remaining sprites were suddenly defined as char 0, transparent and positioned at (0,0)!
-  **CPYBLK** (loaded from FBLOCKS) was copying blocks from previous blocks files if the corresponding blocks were in block buffers. **EMPTY-BUFFERS** was added to fix it.

## Appendix A The fbForth 2.0 Glossary

This appendix corresponds to Appendix D of the manual. The following word descriptions are only those that have been removed, changed or are new to **fbForth 2.0**. As in the manual, they are in ASCII collating order, which is indicated in the footer in this appendix.

Words that have been removed appear in this appendix but have the word and definition completely ~~struck through~~. Those words with changed definitions have only the replaced part of the definition ~~struck through~~. New words, of course, are not struck through but are preceded with a  icon.

### A.1 fbForth 2.0 Word Descriptions

**(ABORT)** Resident

( --- )

Executes after an error when **WARNING** < 0. Normally, **WARNING** = 1. **(ABORT)** normally executes **ABORT**, but may be redirected (with care!) to execute a user's alternative procedure. It is defined as

```
: (ABORT) ABORT ;
```

If you wished to have **(ABORT)** execute your error procedure, say **MY\_ERROR\_PROC**, you would need to replace the CFA of **ABORT** in the definition of **(ABORT)** with the CFA of **MY\_ERROR\_PROC**. Fortunately, this is easy to do! The CFA of **ABORT** sits in the parameter field of **(ABORT)**, the address (PFA) of which is what ticking **(ABORT)** gives you. You can verify this with the following code:

```
HEX ok:0  
' (ABORT) @ U. 6AAC ok:0  
' ABORT CFA U. 6AAC ok:0
```

The second line above ticks **(ABORT)**, fetches the resulting PFA's contents and prints what should be the CFA of **ABORT**. The third line above ticks **ABORT**, gets its CFA and prints it. As you can see, they are, indeed, the same address.

Now, to install your error procedure, simply get its CFA and stash it in the parameter field of **(ABORT)** as follows:

```
' MY_ERROR_PROC CFA ok:1  
' (ABORT) ! ok:0
```

To get your error procedure to run at the next error, set **WARNING** to a negative number as below:

```
-1 WARNING ! ok:0
```

To re-instate normal **fbForth 2.0** error handling, you only need to store a positive number in **WARNING**. You can restore the default action of **(ABORT)** with the following Forth code:

```
' ABORT CFA ok:1  
' (ABORT) ! ok:0
```

**;CODE** *[immediate word]* Resident

( [ ] | CODEh --- )

**;CODE** is now a synonym for **DOES>ASM:** and terminator for **CODE:** .

If **STATE** is “compiling”, **;CODE** executes **DOES>ASM:** , which expects nothing on the stack. In this mode, it is used with **<BUILDS** in the form:

```
      : cccc <BUILDS ... ;CODE <assembly mnemonics> NEXT,
```

Stops compilation and terminates a new defining word, **cccc** , by compiling (**;CODE**) . Sets the **CONTEXT** vocabulary to **ASSEMBLER** , assembling to machine code the assembly mnemonics following **;CODE** .

When **cccc** later executes in the form:

```
      cccc nnnn
```

the word **nnnn** will be created with its execution procedure given by the machine code following (**;CODE**) in the definition of **cccc** , *i.e.*, when **nnnn** is executed, it does so by jumping to that code in **cccc** . An existing defining word (**<BUILDS** in this case) must exist in **cccc** prior to **;CODE** . See Chapter 9 of the manual for more details.

If **STATE** is “executing”, **;CODE** is the terminator for **CODE: cccc ... ;CODE** , which defines a new word **cccc** with machine code contents that do not require the use of , to compile them. It is much faster than compiling with **CODE cccc ... NEXT,** . The following ALC quadruples the value on the stack by double addition:

```
      ASM: QUAD
          *SP *SP A,
          *SP *SP A,
      ;ASM
```

The above code can be re-stated in machine code without requiring the TMS9900 Assembler:

```
      HEX CODE: QUAD A659 A659 ;CODE
```

The equivalent code using **CODE cccc ... NEXT,** is:

```
      HEX CODE QUAD A659 , A659 , NEXT,
```

With a very long definition, using **CODE: cccc ... ;CODE** is significantly faster.

 **>MAP**

Resident

( *bank addr* --- )

This word is ported from TurboForth code courtesy of Mark Wills.

If a SAMS card is present, **>MAP** maps memory bank *bank* to address *addr*.

Address *addr* should be a valid address on a 4KiB boundary, *viz.*, **2000h**, **3000h**, **A000h**, **B000h**, **C000h**, **D000h**, **E000h** or **F000h**. Bank *bank* should be a number between 0 and **FFh**.

It is best to use **S0&TIB!** , *q.v.*, to change **S0** and **TIB** both to **EFA0h** or **DFA0h** (exactly 4KiB or 8KiB [1 or 2 SAMS page(s)] lower than the default **FFA0h** ), thus allowing the use of **E000h** and/or **F000h** with impunity!

When a SAMS memory expansion card is installed, the 32KiB of CPU RAM is actually taken from the SAMS memory. At startup, **fbForth 2.0** reserves the following banks of SAMS memory for the “standard” 32KiB RAM:

<b>Bank</b>	<b>4KiB Boundary</b>
<b>F8h</b>	<b>2000h</b>
<b>F9h</b>	<b>3000h</b>
<b>FAh</b>	<b>A000h</b>
<b>FBh</b>	<b>B000h</b>
<b>FCh</b>	<b>C000h</b>
<b>FDh</b>	<b>D000h</b>
<b>FEh</b>	<b>E000h</b>
<b>FFh</b>	<b>F000h</b>

As can be seen from the above table, fbForth assumes a 1024K SAMS memory card; so, **fbForth 2.0** is not compatible with 256K AMS cards.

Lower RAM **2000h – 3FFFh** is reserved by **fbForth 2.0** for four block buffers, low-level support, system variables and the return stack; therefore, extreme care should be taken when paging banks **F8h** and **F9h** out of **2000h** and **3000h**, respectively. The same care should be taken with upper RAM when paging banks **FAh** and **FFh** out of **A000h** (start of User Dictionary) and **F000h** (TIB and base of parameter stack), respectively.

Because the RAM portion of the dictionary grows up from **A030h** and the parameter stack grows down from **FFA0h**, extreme care must be taken mapping SAMS memory. It is probably advisable to limit SAMS mapping to one or two 4KiB window(s) at **D000h** and/or **E000h**. If **E000h** is used, the stack is limited to 2000 cells, which is probably sufficient for most programming.

**ALIGN**

Resident

( --- )

**ALIGN** insures that **HERE** is on an even address boundary. Use of **C**, is one way **HERE** can land on an odd address boundary. **CREATE** uses **ALIGN** before installing the header for a new word definition. Align is very similar to **=CELLS** except that it neither expects nor leaves anything on the stack.

**ASCII***[immediate word]*

Resident

( --- *ascii* ) ( IS:*token* )

Leaves on the stack the ASCII value of the first character of the next token in the input stream:

**ASCII G . 71 ok:0****ASM>CODE**

ASM&gt;CODE -- Code Output Utility

( --- ) ( IS:*word DSKn.file* )

**ASM>CODE** appends to *DSKn.file* the hexadecimal machine code of a Forth word

written in ALC (Assembly Language Code) in **CODE: word ... ;CODE** format, where ‘...’ represents the machine code in text. This is useful for loading words defined in ALC without the need for loading the **fbForth** TMS9900 Assembler from FBLOCKS. *Please note that **ASM>CODE** should not be used for words in the resident dictionary because word entries in the resident dictionary are in an unconventional, non-contiguous format.*

**ASM>CODE** first checks to insure that *word* is a word defined in ALC. If it is not or it does not exist, **ASM>CODE** quits with an error message to that effect. If it is an ALC word, **ASM>CODE** attempts to open the file *DSKn.file* in Append mode. Failing that, *DSKn.file* is created and opened in Output mode.

As an example you might assemble the word **LDCR**, the ALC for which is listed in Appendix H “Assembly Source for CODEd Words” in the manual, and then run the following code:

```
ASM>CODE LDCR DSK1.CRUWORDS
```

Examining the contents of DSK1.CRUWORDS would reveal the same code as shown in Block #5 of FBLOCKS (17JUN2016 and later).

If you are using the TI-99/4A emulator, *Classic99* ([www.HarmlessLion.com](http://www.HarmlessLion.com)), in Microsoft Windows, you can use the Windows clipboard as the file **CLIP** as follows:

```
ASM>CODE LDCR CLIP
```

See Chapter 9 “The fbForth 2.0 TMS9900 Assembler” in the manual for more information.

**BOOT**

Resident

```
( n | [ ] --- )
```

This word's functionality has been changed from the original TI Forth functionality, which essentially was a continuation of **COLD**. It now simply restarts the system as though the user had just chosen the second or third option on the cartridge menu screen. It expects the default text mode *n* on the stack. The value *n* is forced to 0 or 1 for **TEXT80** or **TEXT**, respectively. **BOOT** may be executed with nothing on the stack, in which case **TEXT** is used.

A key may be held down to select the boot disk number or **<ENTER>** may be held down to prevent loading of FBLOCKS.

**CASE**

```
[immediate word]
```

Resident

Used in a colon definition to initiate the construct:

```
CASE
  n1 OF ... ENDOF
  n2 OF ... ENDOF
  ...
  ELSEOF ... ENDOF <==This clause is optional. See below.
ENDCASE
```

Compile time: ( --- csp 4 )

**CASE** gets the value *csp* of **CSP** to the stack for later restoration at the end of **ENDCASE**'s compile-time activity. It stores the current stack position in **CSP** to help

**ENDCASE** track how many **OF** clause branch distances to process. It finally pushes 4 to the stack for compile-time error checking by **OF** and **ENDCASE** .

RUNTIME: (  $n$  ---  $n$  )

**CASE** itself does nothing with the number  $n$  on the stack; but, it must be there for **OF** , **ELSEOF** or **ENDCASE** to consume. If  $n = n_1$ , the code between the immediately following **OF** and **ENDOF** is executed. Execution then continues after **ENDCASE** . If  $n$  does not match any of the values preceding any **OF** , the code between the last **ENDOF** and **ENDCASE** is executed and may use  $n$ ; but, one cell *must* be left for **ENDCASE** to consume or a stack underflow will result. Execution then continues after **ENDCASE** .

Use of the optional **ELSEOF** obviates the necessity of putting any difficult-to-design default action between the last **ENDOF** and **ENDCASE** .



**CODE:** *[immediate word]* Resident

( --- )

**CODE:** opens a **CODE: cccc ... ;CODE** word definition that employs its own interpreter to quickly convert numbers and compile them. See **;CODE** for a detailed explanation.

**COLD**

Resident

( --- )

**COLD** is the cold-start procedure that may be called from the terminal to remove application programs and to restart **fbForth 2.0**. It is also the last routine executed by the **fbForth 2.0** startup code. Formerly, it was a high-level Forth word that called another high-level Forth word ( **BOOT** ) at its conclusion. They have both been combined into a single ALC routine that (re-)sets the Forth environment to the default startup conditions.

In restarting **fbForth 2.0**, **COLD** resets user variables to their startup values, including the dictionary pointer (to point to just after the resident dictionary), resets the current blocks file to the default DSK $n$ .FBLOCKS ( $n$  is the boot disk number), loads block #1 and executes **ABORT** , *q.v.*

**COLD** may be called from the terminal to remove application programs and to restart **fbForth 2.0**.

See § 2.4 “Changes to COLD” for more detail.



**DATA[** *[immediate word]* Resident

( --- *addr n* ) (IS: $n_1$  ...  $n_n$  )

**DATA[** opens a **DATA[ ... ]DATA** construct that compiles numbers and leaves their beginning address *addr* and cell count  $n$  on the stack. If compiling within another definition, **DATA[** compiles **DATA[ ]** and cell count  $n$  in front of the array.



**DATA[ ]** Resident

( --- *addr n* )

Runtime routine compiled by **DATA[** to push to the stack the address *addr* and number of cells  $n$  of the number array that follows it in a word definition.

**DCHAR**

Resident

*( addr cnt chr --- )*

**DCHAR** is similar to “CALL CHAR” in TI Extended Basic, but is not limited to 4 characters. It is similar to **CHAR**, but uses an array of numbers instead of the stack for pattern definition. It is used to define one or more characters starting at the pattern address of character *chr*. **DCHAR** moves *cnt* cells from address *addr* to the pattern address of character *chr* in VRAM.

**DCT**

Resident

*( --- addr )*

A constant that pushes to the stack the address *addr* of the Default Colors Table for all VDP modes. It also gives the user access to the default text mode because it immediately follows the table.

VDP Mode	Table Offset (bytes)	Screen/Text Colors	Color Table Colors
<b>TEXT80</b>	0	<b>4Fh</b>	<b>00h</b>
<b>TEXT</b>	2	<b>4Fh</b>	<b>00h</b>
<b>GRAPHICS</b>	4	<b>F4h</b>	<b>F4h</b>
<b>MULTI</b>	6	<b>11h</b>	<b>F4h</b>
<b>GRAPHICS2</b>	8	<b>FEh</b>	<b>10h</b>
<b>SPLIT</b>	10	<b>FEh</b>	<b>F4h</b>
<b>SPLIT2</b>	12	<b>FEh</b>	<b>F4h</b>

Default Text Mode	Table Offset (bytes)	VDP Mode
<b>TEXT</b>	14	<b>0001h</b>

All changes to the above values will survive execution of **COLD**.

**DEFBF**

Resident

*( --- addr )*

Gets the address *addr* of the default blocks filename (DSK1.FBLOCKS) in low RAM to the stack. This address points to the string-length byte and can be displayed by

**COUNT TYPE**

If the boot disk is other than DSK1, that will be reflected in the name displayed by the above Forth code.

**ELSEOF***[immediate word]*

Resident

**ELSEOF** is the start of the catchall default **ELSEOF ... ENDOF** clause that occurs inside a colon definition as the optional default clause within the **CASE ... ENDCASE** construct, just before **ENDCASE**. If execution reaches **ELSEOF**, the words between

**ELSEOF** and **ENDOF** will always be executed. There should be no value preceding **ELSEOF** because the runtime stack value will be duplicated in its place to force a match by the compiled (**OF**) .

Use of the **ELSEOF** clause guarantees that **ENDCASE** will never execute. It is a lot easier to use an **ELSEOF** clause instead of trying to contrive a default action ahead of **ENDCASE** . Compare with a description of just such a default action at **CASE** .

Compile time: ( 4 --- *addr* 5 )

Checks for the value 4 on the stack left there by **CASE** or a previous **ENDOF** , compiles **DUP** to force runtime comparison of the value on the stack with itself (guaranteeing a match), compiles (**OF**) , leaves its address *addr* for branching resolution by **ENDOF** and leaves a 5 for its matching **ENDOF** to check.

Runtime: ( *n* *n* --- )

Duplicates the value *n*, which was on top of the stack when **CASE** 's runtime action occurred. Comparison of the two identical numbers forces execution of the words between **ELSEOF** and **ENDOF** . See **CASE** and **ENDOF** .

**ENDCASE** Resident  
*[immediate word]*

Occurs in a colon definition as the termination of the **CASE ... ENDCASE** construct.

Compile time: ( *csp* *addr*<sub>1</sub> ... *addr*<sub>*n*</sub> 4 --- )

It uses the 4 for compile-time error checking. It uses the value in **CSP** put there by **CASE** to track the number of **OF** clauses for which it must calculate branch distances from the addresses (*addr*<sub>1</sub> ... *addr*<sub>*n*</sub>) that each **ENDOF** left on the stack.

Runtime: ( *n* --- )

If all **OF** clauses fail, any code after the last **ENDOF** , including **ENDCASE** , will execute. **ENDCASE** will remove the number *n* left on the stack by the failure of the last **OF** clause.

If you include code between the last **ENDOF** and **ENDCASE** , it must leave at least one number on the stack for **ENDCASE** to consume to prevent stack underflow. See **CASE** .

A better default action is to use an **ELSEOF** clause (with no preceding value) as the last clause before **ENDCASE** . See **ELSEOF** for more information.

**ERROR** Resident

( *n*<sub>1</sub> --- *n*<sub>2</sub> *n*<sub>3</sub> | [ ] )

**ERROR** processes error notification and restarts the interpreter. **WARNING** is first examined. If **WARNING** < 1, (**ABORT**) , *q.v.*, is executed. The sole action of (**ABORT**) is to execute **ABORT** . This allows the user to (*cautiously!*) modify this behavior by replacing the CFA of (**ABORT**) with the CFA of the user's error procedure. **ABORT** clears the stacks and executes **QUIT** , which stops compilation and restarts the interpreter.

If **WARNING** ≥ 0 and the input stream is not the terminal, **ERROR** leaves the contents of **IN** *n*<sub>2</sub> and **BLK** *n*<sub>3</sub> on the stack to assist in determining the location of the error. Execution of **WHERE** , at this point, will open the offending block in the editor and place the cursor at the text immediately following the token that caused the error.

If **WARNING** > 0, **ERROR** prints the error text of system message number  $n_1$ . If **WARNING** = 0, **ERROR** prints  $n_1$  as an error number (This was used in TI Forth in a non-disk installation; but, this is unnecessary in **fbForth 2.0** because the system messages are always present in cartridge ROM). The last thing **ERROR** does is to execute **QUIT**, which, as above, stops compilation and restarts the interpreter.

**FLD** Resident

(~~----addr~~)

~~A user variable for control of number output field width. Presently unused in fbForth and fbForth 2.0.~~

**FNT** Resident

( --- )

**FNT** loads either the default font file (can be changed by user with **USEFFL**, *q.v.*) or the console font into the Pattern Descriptor Table (PDT) depending on the value of the user variable **SCRFONT**. The default font is loaded from DSK1.FBFONT by **FNT** (or from DSK $n$ .FBFONT if key  $n$  is held down) at **fbForth 2.0** startup because **SCRFONT** = -1 at startup. The **fbForth 2.0** system default font contains the patterns for ASCII character codes 0 – 127. The font pattern for each character is 8 bytes, which means that 1KiB of pattern code is loaded into the PDT. This font contains true lowercase characters with true descenders.

Executing **COLD** will maintain the currently selected font as the default. Restarting the system with **BOOT**, **MON** or a power cycle will restore loading of the system font from DSK1.FBFONT.

See Chapter 4 “Screen Font Changes” for more detail.

**ISR** Resident

( --- addr )

A user variable that initially contains 0 to indicate that no user Interrupt Service Routine (ISR) has been installed. The user must modify **ISR** to contain the CFA of the Forth routine to be executed each 1/60 second. Next, the contents of the console ISR hook, **83C4h**, must contain the address of the **fbForth 2.0** ISR, which it does at startup. Note that the interrupt service linkage code address is always available in **INTLNK**.

The console ISR hook, **83C4h**, should be zeroed before changing **ISR** and restored with the value in **INTLNK** after changing it.

See Chapter 3 “Interrupt Service Routines (ISRs)” for much more detail.



**PAGE**

Resident

( --- )

Clears the display screen and places the cursor at the top, left corner. It is a shortcut for

**CLS**  
**0 0 GOTOXY**

**PANEL**

Resident

 $( x y w h \text{ ---} )$ 

Sets up a panel within the video display for **SCROLL** to scroll in any orthogonal direction with or without wrapping, depending on the value of **WRAP**. The panel will be  $w$  characters wide,  $h$  characters high with its upper, left corner at column  $x$  and row  $y$ .

**PLAY**

Resident

 $( addr \text{ flag } \text{ ---} )$ 

This word is ported from TurboForth code courtesy of Mark Wills.

**PLAY** starts the sound list at address  $addr$ , depending on  $flag$ :

Flag	Action
0	Do not play if either sound list is active.
1	Unconditionally play, killing all previous sound lists.
-1	Plays as sound list #2, muting sound list #1 for the duration of sound list #2.

Sound lists consist of a list of sound commands starting with a byte count and ending with a duration count (sixtieths of a second) that is not included in the byte count. The last sound-command list should silence all four sound generators and end with a duration of 0. See Section 20 of the *Editor/Assembler Manual* for details on sound lists.

A sound list may be prepared for **PLAY** with **DATA[ ... ]DATA** by dropping the cell count:

```
DATA[ <sound list> ]DATA
```

```
DROP 1 PLAY
```

**PLAYING?**

Resident

 $( \text{ --- } \text{ flag } )$ 

This word is ported from TurboForth code courtesy of Mark Wills.

**PLAYING?** checks both **fbForth 2.0** sound status registers, ORs them and leaves that value on the stack as  $flag$ . If  $flag = 0$ , no sound list is active.

**PLAYING?** is intended for use with **PLAY**, not **SOUND**. **SOUND** does not use the **fbForth 2.0** sound status registers.

**RP@**

Resident

 $( \text{ --- } \text{ addr } )$ 

Returns the address  $addr$  of the current top of the return stack.

**S"** $[ \text{ immediate word } ]$ 

Resident

 $( \text{ --- } \text{ addr } [ [ ] ] ( \text{ IS: string" } )$ 

Accepts a string from the input stream (IS) until ‘’ is encountered. When executing,

the packed string is stored at **PAD** and the address *addr* of the length byte is left on the stack.

When compiling a word definition, **SLIT** is first compiled into the definition, then the packed string. Later, when the word is executed, **SLIT** will push the address of the string's length byte to the stack and skip over the string to the word following it in the definition.

**S0** Resident

( --- *addr* )

User variable that points to the base of the parameter stack. Pronounced "s zero". See **SP!** .



**S0&TIB!** Resident

( *addr*<sub>1</sub> --- *addr*<sub>2</sub> )

This word is primarily for use in a 1024-byte SAMS environment, where it is or may be necessary to move the stack base and TIB buffer, both of which start up at the same address, *viz.*, **FFA0h**. **S0&TIB!** forces *addr*<sub>1</sub> to **AFA0h**, **BFA0h**, **CFA0h**, **DFA0h**, **EFA0h** or **FFA0h**; copies it to the User Variables, **S0** and **TIB** , in the table of default values so the settings will survive **COLD** ; and leaves the new address on the stack as *addr*<sub>2</sub>. The lower limit is forced above **HERE** so as not to destroy the user's dictionary.



**SAMS!** Resident

( --- )

This word is ported from TurboForth code courtesy of Mark Wills.

This calls the SAMS initialization in the startup code in bank 1 to restore SAMS mapping to initial conditions.



**SAMS?** Resident

( --- *flag* )

This word is ported from TurboForth code courtesy of Mark Wills.

Leaves a copy of the SAMS flag from startup as *flag*.



**SAY** Resident

( *addr* *n* --- )

This word is ported from TurboForth code courtesy of Mark Wills.

**SAY** needs on the stack the address *addr* of a block of Speech Synthesizer ROM speech addresses and the number *n* of those addresses. This can be accomplished with **DATA[ ... ]DATA** . Consult Section 22 of the *Editor/Assembler Manual* for details.

**SCRFONT** Resident

( --- *addr* )

A user variable containing a flag indicating whether **FNT** should load the current default font (flag ≠ 0) or the console font (flag = 0). Changing the value in **SCRFONT**

does not take effect until the next time **FNT** is executed.

See Chapter 4 “Screen Font Changes” for more detail.



## SCROLL

Resident

( *dir* --- )

Scrolls the display screen panel set up by **PANEL** in direction *dir*. **PANEL** must be executed at least once before **SCROLL** because its parameters are indeterminate after powerup. Acceptable values for *dir* are

<u>Direction</u>	<u>Value</u>
left	0
right	2
up	4
down	6



## SOUND

Resident

( *pitch vol ch#* --- )

This word is ported from TurboForth code courtesy of Mark Wills.

Pitch *pitch*, volume *vol* and channel *ch#* are as described in the *Editor/Assembler Manual* in Section 20. Pitch values range from 0 – 1023, 0 representing the highest pitch. Volume values range from 0 – 15, 15 representing silence. Channels 0 – 2 represent the corresponding tone generators and channel 3 is the noise generator.

**SOUND** uses the pitch value for setting the type of noise for the noise generator (channel 3). Shift rates are 0 – 3. Noise type can be white noise (0) or periodic noise (4). The pitch value to pass to **SOUND** is the sum of shift rate and noise type and ranges from 0 – 7.

Once a tone or noise generator is started, the sound/noise continues until silenced by executing **SOUND** with a volume of 15. The pitch must be supplied, but is irrelevant. The following Forth code will silence channel 2:

```
0 15 2 SOUND
```



## SPDCHAR

Resident

( *addr cnt chr* --- )

Same as **DCHAR**, but for sprite pattern definitions because SPDTAB does not always start at the same VRAM address as PDT.



## STREAM

Resident

( *addr n* --- )

This word is ported from TurboForth code courtesy of Mark Wills.

**STREAM** needs on the stack the address *addr* of a block of raw speech data to be spoken and the number of cells *n* in the buffer. This can be accomplished with **DATA[ ... ]DATA**. **STREAM** will feed the raw speech data to the Speech Synthesizer.

-  **TALKING?** Resident
- ( --- *flag* )
- This word is ported from TurboForth code courtesy of Mark Wills.
- TALKING?** returns *flag* = 0 if the Speech Synthesizer is idle, otherwise, *flag* = 1.
- It is a good idea to use **TALKING?** to insure the Speech Synthesizer is not busy before executing **SAY** or **STREAM** .
-  **TIF2FBF** TI Forth Block Utilities
- ( IS:*srcStrtBlk srcEndBlk DSKn dstStrtBlk dstFile* )
- Copies the range of blocks (screens) *srcStrtBlk* – *srcEndBlk* from TI Forth disk *DSKn* to **fbForth** blocks file *dstFile*, starting at block *dstStrtBlk*.
-  **TIFBLK** TI Forth Block Utilities
- ( IS:*blk DSKn* )
- Lists block (screen) *blk* of TI Forth disk *DSKn* to the display. The display will pause for user intervention in Text mode due to wrapping 64-byte lines on a 40-column display.
-  **TIFIDX** TI Forth Block Utilities
- ( IS:*strtBlk endBlk DSKn* )
- Lists the index (line #0) lines of a range of blocks (screens) *strtBlk* – *endBlk* of TI Forth disk *DSKn* to the display. The display will pause for user intervention if the list requires scrolling.
-  **TIFVU** TI Forth Block Utilities
- ( IS:*blk DSKn* )
- Starts the TI Forth disk browser/copier at block (screen) *blk* of TI Forth disk *DSKn*. The browser is patterned after the **fbForth** block editors, allowing scrolling left and right by panels and blocks. The user may also copy a range of TI Forth blocks to an **fbForth** blocks file, which must have been created prior to entering the browser/copier.
-  **TOKEN** Resident
- ( *delim* --- *addr* | [ ] ) ( IS:*string*" )
- TOKEN** gets a string ending with *delim* from the input stream (IS) into **PAD** as a packed string and passes the address *addr* of the string's length byte on the stack if interpreting (command line or loading), but compiles the packed string to **HERE** , with nothing to the stack, if compiling.
- TOKEN** is used by several words in the resident dictionary, including **MKBFL** , **USEBFL** , **S"** , **."** , **WLITERAL** and **USEFFL** .

**VLIST** \*\*\**Definition is the same, but now ==>* Resident

( --- )

Prints the names of all words defined in the **CONTEXT** vocabulary. Note that **VLIST** will display the names of even ill-defined words in the dictionary that cannot be found with **'**, **-FIND** or **(FIND)**, *q.v.*, because their smudge bits are set. See **SMUDGE** and **PAUSE**.

**WARNING** Resident

( --- *addr* )

A user variable (initialized by **COLD** to 1 at system startup), containing a value controlling messages.

If **WARNING** > 0, full-text system error messages are displayed by **MESSAGE** and **ERROR**, which executes **MESSAGE**.

If **WARNING** = 0, messages will be presented by number ( **msg #n** ). In TI Forth, it means the disk is unavailable; but, this is not necessary in **fbForth 2.0** because error messages are always memory resident.

If **WARNING** < 0 when **ERROR** executes, **ERROR** will execute **(ABORT)**, which can be redefined to execute a user-specified procedure instead of the default **ABORT**.

See **MESSAGE**, **(ABORT)**, **ERROR** and **?ERROR** for more detail.



**WRAP** Resident

( --- *addr* )

A user variable containing the wrapping flag for **SCROLL**. A non-zero value signals **SCROLL** to wrap the disappearing row or column of the panel set up by **PANEL** to the opposite side of the panel. The initial value of **WRAP** is 0.



**[DCHAR]** Resident

( *addr cnt chr vaddr* --- )

Helper routine for **DCHAR** and **SPDCHAR**.



**]DATA** Resident *[immediate word]*

( --- )

**]DATA** closes a **DATA[ ... ]DATA** construct that compiles numbers and leaves their beginning address and cell count on the stack. If compiling within another definition, **]DATA** stores the cell count between the compiled **DATA[ ]** and the first number of the array.

## Appendix B User Variables in fbForth 2.0

This appendix corresponds to Appendix F of the manual. There is one new word ( **WRAP** ) added to this table. There is one word ( **FLD** ) no longer part of this table.

The complete table is included here for context. See the corresponding appendix of the manual for more detail.

### B.1 fbForth 2.0 User Variables (Address Offset Order)

Name	Offset	Initial Value	Description
UCONS\$	06h	366Ch	Base of User Var initial value table
S0	08h	FFA0h	Base of Stack
R0	0Ah	3FFEh	Base of Return Stack
U0	0Ch	36B6h	Base of User Variables
TIB	0Eh	FFA0h	Terminal Input Buffer address
WIDTH	10h	31	Name length in dictionary
DP	12h	A000h	Dictionary Pointer
SYS\$	14h	30DEh	Address of System Support
CURPOS	16h	0	Cursor location in VDP RAM
INTLNK	18h	3020h	Pointer to Interrupt Service Linkage
WARNING	1Ah	1	Message Control
C/L\$	1Ch	64	Characters per Line
FIRST\$	1Eh	2010h	Beginning of Disk Buffers
LIMIT\$	20h	3020h	End of Disk Buffers
COLTAB	22h	380h	Color Table address in VRAM. <b>COLTAB</b> gets addr.
SATR	24h	300h	Sprite Attribute Table address in VRAM. <b>SATR</b> gets addr.
SMTN	26h	780h	Sprite Motion Table address in VRAM. <b>SMTN</b> gets addr.
PDT	28h	800h	Pattern Descriptor Table addr in VRAM. <b>PDT</b> gets addr.
FPB	2Ah	80h	User font file PAB offset from FRB. <b>FPB</b> gets addr.
DISK_BUF	2Ch	1000h	VDP location of 128B Forth Record Buffer (FRB)
PABS	2Eh	460h	VDP location for PABs
SCRN_WIDTH	30h	40	Display Screen Width in Characters
SCRN_START	32h	0	Display Screen Image Start in VDP
SCRN_END	34h	960	Display Screen Image End in VDP
ISR	36h	 0	Interrupt Service Pointer
ALTIN	38h	0	Alternate Input Pointer
ALTOUT	3Ah	0	Alternate Output Pointer
VDPMDE	3Ch	1	VDP Mode
BPB	3Eh	C6h	Blocks PABs offset from FRB. <b>BPB</b> gets address.
BPOFF	40h	0	Current Blocks file offset from <b>BPB</b> . (0 or 70h)
SPDTAB	42h	800h	Sprite Descriptor Table addr in VRAM. <b>SPDTAB</b> gets addr.
SCRFONT	44h	 -1	Flag for default/user font (≠ 0) or console font (= 0)
JMODE	46h	0	Flag for whether <b>JOYST</b> executes <b>JKBD</b> (=0) or <b>JCRU</b> (≠0)

Name	Offset	Initial Value	Description
 <b>WRAP</b>	48h	0	Flag for no wrap (= 0) or wrap ( $\neq 0$ ); used by <b>SCROLL</b>
<b>FENCE</b>	4Ah		Dictionary Fence
<b>BLK</b>	4Ch		Block being interpreted
<b>IN</b>	4Eh		Byte offset in text buffer
<b>OUT</b>	50h		Incremented by <b>EMIT</b>
<b>SCR</b>	52h		Last Forth Block (Screen) referenced
<b>CONTEXT</b>	54h		Pointer to Context Vocabulary
<b>CURRENT</b>	56h		Pointer to Current Vocabulary
<b>STATE</b>	58h		Compilation State
<b>BASE</b>	5Ah		Number Base for Conversions
<b>DPL</b>	5Ch		Decimal Point Location
<b>FLD</b>	5Ch		<del>Field Width (unused)</del>
<b>CSP</b>	5Eh		Stack Pointer for error checking
<b>R#</b>	60h		Editing Cursor location
<b>HLD</b>	62h		Holds address during numeric conversion
<b>USE</b>	64h		Next Block Buffer to Use
<b>PREV</b>	66h		Most recently accessed disk buffer
<b>ECOUNT</b>	68h		Error control
<b>VOC-LINK</b>	6Ah		Vocabulary linkage
[user to define]	6Ch		—available to user—
[user to define]	6Eh		—available to user—
[user to define]	70h		—available to user—
[user to define]	72h		—available to user—
[user to define]	74h		—available to user—
[user to define]	76h		—available to user—
[user to define]	78h		—available to user—
[user to define]	7Ah		—available to user—
[user to define]	7Ch		—available to user—
[user to define]	7Eh		—available to user—

## B.2 fbForth 2.0 User Variables (Variable Name Order)

Name	Offset	Initial Value	Description
ALTIN	38h	0	Alternate Input Pointer
ALTOUT	3Ah	0	Alternate Output Pointer
BASE	5Ah		Number Base for Conversions
BLK	4Ch		Block being interpreted
BPB	3Eh	06h	Blocks PABs offset from FRB. <b>BPB</b> gets address.
BPOFF	40h	0	Current Blocks file offset from <b>BPB</b> . (0 or 70h)
C/L\$	1Ch	64	Characters per Line
COLTAB	22h	380h	Color Table address in VRAM. <b>COLTAB</b> gets addr.
CONTEXT	54h		Pointer to Context Vocabulary
CSP	5Eh		Stack Pointer for error checking
CURPOS	16h	0	Cursor location in VDP RAM
CURRENT	56h		Pointer to Current Vocabulary
DISK_BUF	2Ch	1000h	VDP location of 128B Forth Record Buffer (FRB)
DP	12h	A000h	Dictionary Pointer
DPL	5Ch		Decimal Point Location
ECOUNT	68h		Error control
FENCE	4Ah		Dictionary Fence
FIRST\$	1Eh	2010h	Beginning of Disk Buffers
FLD	5Ch		Field Width (unused)
FPB	2Ah	80h	User font file PAB offset from FRB. <b>FPB</b> gets addr.
HLD	62h		Holds address during numeric conversion
IN	4Eh		Byte offset in text buffer
INTLNK	18h	3020h	Pointer to Interrupt Service Linkage
ISR	36h	 0	Interrupt Service Pointer
JMODE	46h	0	Flag for whether <b>JOYST</b> executes <b>JKBD</b> (=0) or <b>JCRU</b> (≠0)
LIMIT\$	20h	3020h	End of Disk Buffers
OUT	50h		Incremented by <b>EMIT</b>
PABS	2Eh	460h	VDP location for PABs
PDT	28h	800h	Pattern Descriptor Table addr in VRAM. <b>PDT</b> gets addr.
PREV	66h		Most recently accessed disk buffer
R#	60h		Editing Cursor location
R0	0Ah	3FFEh	Base of Return Stack
S0	08h	FFA0h	Base of Stack
SATR	24h	300h	Sprite Attribute Table address in VRAM. <b>SATR</b> gets addr.
SCR	52h		Last Forth Block (Screen) referenced
SCRFNT	44h	 -1	Flag for default/user font (≠ 0) or console font (= 0)
SCRN_END	34h	960	Display Screen Image End in VDP
SCRN_START	32h	0	Display Screen Image Start in VDP
SCRN_WIDTH	30h	40	Display Screen Width in Characters
SMTN	26h	780h	Sprite Motion Table address in VRAM. <b>SMTN</b> gets addr.
SPDTAB	42h	800h	Sprite Descriptor Table addr in VRAM. <b>SPDTAB</b> gets addr.
STATE	58h		Compilation State
SYS\$	14h	30DEh	Address of System Support

Name	Offset	Initial Value	Description
<b>TIB</b>	<b>0Eh</b>	<b>FFA0h</b>	Terminal Input Buffer address
<b>U0</b>	<b>0Ch</b>	<b>36B6h</b>	Base of User Variables
<b>UCONS\$</b>	<b>06h</b>	<b>366Ch</b>	Base of User Var initial value table
<b>USE</b>	<b>64h</b>		Next Block Buffer to Use
<b>VDPMDE</b>	<b>3Ch</b>	<b>1</b>	VDP Mode
<b>VOC-LINK</b>	<b>6Ah</b>		Vocabulary linkage
<b>WARNING</b>	<b>1Ah</b>	<b>1</b>	Message Control
<b>WIDTH</b>	<b>10h</b>	<b>31</b>	Name length in dictionary
 <b>WRAP</b>	<b>48h</b>	<b>0</b>	Flag for no wrap (= 0) or wrap ( $\neq 0$ ); used by <b>SCROLL</b>
[user to define]	<b>7Eh</b>		—available to user—
[user to define]	<b>7Ch</b>		—available to user—
[user to define]	<b>7Ah</b>		—available to user—
[user to define]	<b>78h</b>		—available to user—
[user to define]	<b>76h</b>		—available to user—
[user to define]	<b>74h</b>		—available to user—
[user to define]	<b>72h</b>		—available to user—
[user to define]	<b>70h</b>		—available to user—
[user to define]	<b>6Eh</b>		—available to user—
[user to define]	<b>6Ch</b>		—available to user—

## Appendix C fbForth 2.0 Load Option Directory

This appendix corresponds to Appendix G of the manual. New options are so indicated.

The load options are displayed by typing **MENU**. The load options allow you to load only the Forth extensions you wish to use.

You will notice that some of the load options first load other Forth blocks upon which they depend. For example, option, 64-Column Editor, depends on the words loaded by block 13, which displays “loading compact list words” as block 13 starts to load. If, by chance, the prerequisite words were already in the dictionary at the time you type **6 LOAD**, they would not be loaded again. This is called a conditional load.

### ***C.1 Option: 64-Column Editor***

Starting screen: 6

Words loaded: **EDIT**            **ED@**            **WHERE**  
                   **CLIST**            **CLINE**

### ***C.2 Option: CPYBLK -- Block Copying Utility***

Starting screen: 4

Words loaded: **SCMP**            **CPYBLK**

### ***C.3 Option: Memory Dump Utility***

Starting screen: 16

Words loaded: **DUMP**

### ***C.4 Option: TRACE -- Colon Definition Tracing***

Starting screen: 18

Words loaded: **TRACE**            **UNTRACE**        **TRON**  
                   **TROFF**            : *(alternate)*

### ***C.5 Option: Printing Routines***

Starting screen: 19

Words loaded: **SWCH**            **UNSWCH**        **?ASCII**  
                   **TRIAD**            **TRIADS**        **INDEX**

### **C.6 Option: TMS9900 Assembler**

Starting screen: 21

Words loaded: Entire Assembler vocabulary. See Chapter 9 of the manual.

### **C.7 Option: CRU Words**

Starting screen: 5

Words loaded: **SB0**            **SBZ**            **TB**  
                   **LDCR**            **STCR**

### **C.8 Option: More Useful Stack Words etc.**

Starting screen: 41

Words loaded: **2DUP**            **2DROP**            **NIP**            **TUCK**            **CELLS**            **-ROT**  
                   **PICK**            **ROLL**            **WITHIN**            **<>**            **\$.**            **EXIT**

### **C.9 Option: Stack-based String Library**

Starting Screen: 42

Words loaded: Entire String Stack Library. See Chapter 14 of the manual.

### **C.10 Option: DIR -- Disk Catalog Utility**

Starting screen: 36

Words loaded: **DIR**

### **C.11 Option: CAT -- Disk Catalog Utility**

Starting screen: 58

Words loaded: **CAT**

### **C.12 Option: TI Forth Block Utilities**

Starting screen: 61

Words loaded: **TIFBLK**            **TIFIDX**            **TIF2FBF**            **TIFVU**

## Appendix D Contents of FBLOCKS

This appendix corresponds with Appendix J of the manual.

The contents of the **fbForth 2.0** system blocks file, FBLOCKS, that follow are derived from TI Forth but are in different blocks. Much of this is due to the fact that the blocks are in a file rather than referenced as sectors on a disk. The blocks are also not necessarily in the same order as in TI Forth; however, the TI Forth block (screen) number is indicated as “(old TIF #...)” where applicable. There are also many changes from TI Forth. Many words have been moved to the resident dictionary and some TI Forth words have been removed. There are new words in **fbForth 2.0**, as well. (*cf.* Appendix E in the manual)

Note that blocks are numbered from 1 in **fbForth 2.0** rather than 0 as in TI Forth. There are also 15 blank blocks (blocks 14, 57, 68 – 80), which you can use as you wish. Note, also, that the following file is dated 20JUN2016.

### BLOCK #1 ( old TIF #3)

```

0 ( fbForth WELCOME SCREEN---LES 20JUN2016)
1 BASE->R HEX
2 : MENU 1 BLOCK 2+ @ 6662 - 5 ?ERROR 2 LOAD ;
3 ." FBLOCKS mod: 20JUN2016"
4 CR CR ." Type MENU for load options." CR CR R->BASE ;S
5
6
7
8
9
10
11
12
13
14
15

```

### BLOCK #2

```

0 PAGE ." Load Options (20JUN2016) fbForth 2.0:"
1 ( Type build #) BASE->R HEX 6033 C@ EMIT R->BASE CR CR
2 ." Description Load Block" CR
3 ." -----" CR
4 ." CPYBLK -- Block Copying Utility.....4" CR
5 ." CRU Words.....5" CR
6 ." 64-Column Editor.....6" CR
7 ." Memory Dump Utility.....16" CR
8 ." TRACE -- Colon Definition Tracing....18" CR
9 ." Printing Routines.....19" CR
10 ." TMS9900 Assembler.....21" CR
11 ." More Useful Stack Words etc.....41" CR
12 ." Stack-based String Library.....42" CR
13 ." DIR -- Disk Catalog Utility.....36" CR
14 ." CAT -- Disk Catalog Utility.....58" CR
15 ." TI Forth Block Utilities.....61" CR

```

-->

## BLOCK #3

```

0 ." ASM>CODE -- Code Output Utility.....39" CR
1 ." TMS9900 Assembler (v2.0:8 binary)....27" CR
2 ." 64-Column Editor (v2.0:8 binary).....32" CR
3 ." String Library (v2.0:8 binary).....52" CR CR
4 ." Type <block> LOAD to load. " ;S
5
6
7
8
9
10
11
12
13
14
15

```

## BLOCK #4 ( old TIF #39)

```

0 ( Block Copy 17JUN2016 LES ) CR CR ." CPYBLK copies a range
1 of blocks to the same or another file, e.g.," CR CR ." CPYB
2 LK 5 8 DSK1.F1 9 DSK2.F2" CR CR ." will copy blocks 5-8 from DS
3 K1.F1 to DSK2.F2 starting at block 9." CR CR 0 CLOAD CPYBLK
4 BASE->R DECIMAL 0 VARIABLE SFL 0 VARIABLE DFL 0 CONSTANT XD
5 : SCMP OVER C@ OVER C@ OVER OVER - SGN >R MIN 1+ 0 SWAP 1 DO
6 DROP OVER I + C@ OVER I + C@ - SGN DUP IF LEAVE THEN LOOP R>
7 OVER 0= IF OR ELSE DROP THEN SWAP DROP SWAP DROP ; : GNUM BL
8 WORD HERE NUMBER DROP ; : GBFL BL WORD HERE DUP C@ 1+ =CELLS
9 ALLOT SWAP ! ; : CPYBLK EMPTY-BUFFERS 1 ' XD ! HERE BPB BPOFF
10 @ + 9 + DUP VSBR 1+ HERE SWAP DUP =CELLS ALLOT VMBR GNUM GNUM
11 OVER OVER > IF SWAP THEN OVER - 1+ >R SFL GBFL GNUM DFL GBFL SFL
12 @ DFL @ SCMP 0= IF OVER OVER - DUP 0< SWAP R MINUS > + 2 = IF
13 SWAP R + 1- SWAP R + 1- -1 ' XD ! THEN THEN CR R> 0 DO OVER DUP
14 . OVER SFL @ (UB) SWAP BLOCK 2- ! DFL @ (UB) UPDATE FLUSH XD +
15 SWAP XD + SWAP LOOP DROP DROP DUP (UB) DP ! ; R->BASE

```

## BLOCK #5 ( old TIF #88)

```

0 ( CRU WORDS 12OCT82 LA0 ) 0 CLOAD STCR
1 CR ." loading CRU words"
2 BASE->R HEX
3 CODE: SB0 C339 A30C 1D00 ;CODE
4 CODE: SBZ C339 A30C 1E00 ;CODE
5 CODE: TB C319 A30C 04D9 1F00 1601 0599 ;CODE
6
7 CODE: LDCR C339 A30C C079 C039 0241 000F 1304 0281
8 0008 1501 06C0 0A61 0261 3000 0481 ;CODE
9
10 CODE: STCR C339 A30C C059 04C0 0241 000F C081 0A61 0261 3400
11 0481 C082 1304 0282 0008 1501 06C0 C640 ;CODE
12
13 CR ." See Manual for usage." CR R->BASE
14
15

```

```

BLOCK #6 ( old TIF #22)
0 ( 64 COLUMN EDITOR )
1 0 CLOAD EDITOR2 ( ED@)
2 BASE->R DECIMAL 13 R->BASE CLOAD CLIST
3 BASE->R HEX CR ." loading 64-column editor"
4
5
6 VOCABULARY EDITOR2 IMMEDIATE EDITOR2 DEFINITIONS
7 0 VARIABLE CUR
8 : !CUR 0 MAX 3FF MIN CUR ! ;
9 : +CUR CUR @ + !CUR ;
10 : +LIN CUR @ C/L / + C/L * !CUR ; DECIMAL
11 : LINE. DO I SCR @ (LINE) I CLINE LOOP ;
12
13 : PTR CUR @ SCR @ BLOCK + ;
14 : R/C CUR @ C/L /MOD ; ( --- col row ) R->BASE -->
15

BLOCK #7 ( old TIF #23)
0 ( 64 COLUMN EDITOR ) BASE->R HEX ." ."
1
2 : CINIT
3 SATR 2 0 DO DUP >R D000 SP@ R> 2 VMBW DROP 4 + LOOP DROP
4 0000 0000 0000 0000 5 SPCHAR 0 CUR !
5 F090 9090 9090 90F0 6 SPCHAR 0 1 F 5 0 SPRITE ; DECIMAL
6
7 : PLACE CUR @ 64 /MOD 8 * 1+ SWAP 4 * 1- DUP 0< IF DROP 0 ENDIF
8 SWAP 0 SPRPUT ;
9 : UP -64 +CUR PLACE ;
10 : DOWN 64 +CUR PLACE ;
11 : LEFT -1 +CUR PLACE ;
12 : RIGHT 1 +CUR PLACE ;
13 : CGOTOXY ( col row --- ) 64 * + !CUR PLACE ;
14
15 R->BASE -->

BLOCK #8 ( old TIF #24)
0 ( 64 COLUMN EDITOR ) BASE->R ." ."
1
2 DECIMAL
3
4 : .CUR CUR @ C/L /MOD CGOTOXY ;
5 : DELHALF PAD 64 BLANKS PTR PAD C/L R/C DROP - CMOVE ;
6
7 : DELLIN R/C SWAP MINUS +CUR PTR PAD C/L CMOVE DUP L/SCR SWAP
8 DO PTR 1 +LIN PTR SWAP C/L CMOVE LOOP
9 0 +LIN PTR C/L 32 FILL C/L * !CUR ;
10 : INSLIN R/C SWAP MINUS +CUR L/SCR +LIN DUP 1+ L/SCR 0 +LIN
11 DO PTR -1 +LIN PTR SWAP C/L CMOVE -1 +LOOP
12 PAD PTR C/L CMOVE C/L * !CUR ;
13 : RELINE R/C SWAP DROP DUP LINE. UPDATE .CUR ;
14 : +.CUR +CUR .CUR ;
15 R->BASE -->

```

```

BLOCK #9 ( old TIF #25)
0 ( 64 COLUMN EDITOR ) BASE->R DECIMAL ." ."
1 : -TAB PTR DUP C@ BL >
2 IF BEGIN 1- DUP -1 +CUR C@ BL =
3 UNTIL
4 ENDIF
5 BEGIN CUR @ IF 1- DUP -1 +CUR C@ BL > ELSE .CUR 1 ENDIF UNTIL
6 BEGIN CUR @ IF 1- DUP -1 +CUR C@ BL = DUP IF 1 +.CUR ENDIF
7 ELSE .CUR 1 ENDIF
8 UNTIL DROP ;
9 : TAB PTR DUP C@ BL = 0=
10 IF BEGIN 1+ DUP 1 +CUR C@ BL =
11 UNTIL
12 ENDIF
13 CUR @ 1023 = IF .CUR 1
14 ELSE BEGIN 1+ DUP 1 +CUR C@ BL > UNTIL .CUR
15 ENDIF DROP ; R->BASE -->

BLOCK #10 ( old TIF #26)
0 ( 64 COLUMN EDITOR ) BASE->R ." ."
1 DECIMAL
2 : !BLK PTR C! UPDATE ;
3 : BLNKS PTR R/C DROP C/L SWAP - 32 FILL ;
4 : HOME 0 0 CGOTOXY ;
5 : REDRAW SCR @ CLIST UPDATE .CUR ;
6 : SCRNO CLS 0 0 GOTOXY ." BLOCK #" SCR @ BASE->R DECIMAL U.
7 R->BASE CR ;
8 : +SCR SCR @ 1+ DUP SCR ! SCRNO CLIST ;
9 : -SCR SCR @ 1- 1 MAX DUP SCR ! SCRNO CLIST ;
10 : DEL PTR DUP 1+ SWAP R/C DROP C/L SWAP - CMOVE 32
11 PTR R/C DROP - C/L + 1- C! ;
12 : INS 32 PTR DUP R/C DROP C/L SWAP - + SWAP DO
13 I C@ LOOP DROP PTR DUP R/C DROP C/L SWAP - + 1- SWAP 1- SWAP
14 DO I C! -1 +LOOP ; R->BASE -->
15

BLOCK #11 ( old TIF #27)
0 ( 64 COLUMN EDITOR 15JUL82 LA0 ) BASE->R DECIMAL ." ."
1 0 VARIABLE BLINK 0 VARIABLE OKEY
2 10 CONSTANT RL 150 CONSTANT RH 0 VARIABLE KC RH VARIABLE RLOG
3 : RKEY BEGIN ?KEY -DUP 1 BLINK +! BLINK @ DUP 60 < IF 6 0 SPRPAT
4 ELSE 5 0 SPRPAT ENDIF 120 = IF 0 BLINK ! ENDIF
5 IF ( SOME KEY IS PRESSED ) KC @ 1 KC +! 0 BLINK !
6 IF ( WAITING TO REPEAT ) RLOG @ KC @ <
7 IF ( LONG ENOUGH ) RL RLOG ! 1 KC ! 1 ( FORCE EXT)
8 ELSE OKEY @ OVER =
9 IF DROP 0 ( NEED TO WAIT MORE )
10 ELSE 1 ( FORCE EXIT ) DUP KC ! ENDIF
11 ENDIF
12 ELSE ( NEW KEY ) 1 ( FORCE LOOP EXIT ) ENDIF
13 ELSE ( NO KEY PRESSED) RH RLOG ! 0 KC ! 0
14 ENDIF
15 UNTIL DUP OKEY ! ; R->BASE -->

```

```

BLOCK #12 ( old TIF #28 & 29)
0 ( 64 COLUMN EDITOR )                BASE->R HEX      ." ."
1 : EDT  VDPME @ >R  SPLIT ( 0 1000 040 VFILL) ( 0F 7 VWTR)
2 ( 1000 800 01B VFILL) CINIT !CUR R/C CGOTOXY
3 DUP DUP SCR ! SCRNO CLIST BEGIN RKEY CASE 08 OF LEFT ENDOF
4 0C OF -SCR ENDOF 0A OF DOWN ENDOF 03 OF DEL RELINE ENDOF
5 0B OF UP ENDOF 04 OF INS RELINE ENDOF 09 OF RIGHT ENDOF
6 07 OF DELLIN REDRAW ENDOF 06 OF INSLIN REDRAW ENDOF
7 0E OF HOME ENDOF 02 OF +SCR ENDOF 16 OF TAB ENDOF
8 0D OF 1 +LIN .CUR PLACE ENDOF 1E OF INSLIN BLNKS REDRAW ENDOF
9 01 OF DELHALF BLNKS RELINE ENDOF 7F OF -TAB ENDOF
10 0F OF 5 0 SPRPAT R> VMODE CLS SCRNO DROP QUIT ENDOF
11 DUP 1F > OVER 7F < AND IF DUP !BLK R/C SWAP DROP DUP SCR @
12 (LINE) ROT CLINE 1 +.CUR ELSE 7 EMIT ENDIF ENDCASE AGAIN ;
13 FORTH DEFINITIONS : EDIT EDITOR2 0 EDT ;
14 : WHERE EDITOR2 SWAP 2- EDT ; : ED@ EDITOR2 SCR @ SCRNO EDIT ;
15 CR CR ." See Manual for usage." CR R->BASE

```

```

BLOCK #13 ( old TIF #65 - #66)
0 ( COMPACT LIST )
1 0 CLOAD CLIST BASE->R CR ." loading compact list words"
2 DECIMAL 0 VARIABLE TCHAR 382 ALLOT
3 15 BLOCK 192 + TCHAR 384 CMOVE HEX
4 TCHAR 7C - CONSTANT TC 0 VARIABLE BADDR 0 VARIABLE INDX
5 0 VARIABLE LB FE ALLOT
6 CODE: SMASH ( ADDR #CHAR LINE# --- LB VADDR CNT )
7 C079 C0B9 C0D9 0204 LB , C644 0649 06C1 0221 2000 C641 C042
8 0581 0241 FFFE 0649 0A21 C641 A083 80C2 1501 1020 04C5 04C6
9 D173 D1B3 0965 0966 C025 TC , C066 TC , 0B41 020C 0004 C2C0
10 024B F000 C1C1 0247 0F00 E1CB DD07 0BC0 0BC1 060C 16F4 05C5
11 05C6 C305 024C 0002 16E7 10DD ;CODE
12 DECIMAL
13 : CLINE LB 100 ERASE SMASH VMBW ;
14 : CLOOP DO I 64 * OVER + 64 I CLINE LOOP DROP ;
15 : CLIST BLOCK 16 0 CLOOP ; R->BASE

```

BLOCK #14

```

0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

```

**BLOCK #15 ( old TIF #67)**

```

0 ( Tiny character patterns for TCHAR array---compact list for
1 64-column editor---388 bytes, lines 3:0-9:0 below )
2
3
4
5
6
7
8
9
10
11
12
13
14
15

```

**BLOCK #16 ( old TIF #42)**

```

0 ( DUMP ROUTINES 12JUL82 LCT...25OCT2015 LES mod)
1 0 CLOAD DUMP BASE->R HEX CR ." loading memory dump utility"
2 : VM+ VDPMD @ 0= IF + ELSE DROP THEN ;
3 : DUMP8 -DUP
4 IF
5 BASE->R HEX 0 OUT ! OVER 4 U.R 3A EMIT
6 OVER OVER 0 DO
7 DUP @ 0 <# # # # BL HOLD BL HOLD #> TYPE 2+ 2
8 +LOOP DROP 1F 18 VM+ OUT @ - SPACES
9 0 DO
10 DUP C@ DUP 20 < OVER 7E > OR
11 IF DROP 2E ENDIF
12 EMIT 1+
13 LOOP
14 CR R->BASE ENDIF ; -->
15

```

**BLOCK #17 ( old TIF #43)**

```

0 ( DUMP ROUTINES 12JUL82 LCT...25OCT2015 LES mod) ." ."
1 : DUMP CR 00 8 8 VM+ U/ >R SWAP R> -DUP
2 IF 0
3 DO 8 8 VM+ DUMP8 PAUSE IF SWAP DROP 0 SWAP LEAVE ENDIF LOOP
4 ENDIF SWAP DUMP8 DROP ;
5 ( .S and VLIST have been put in resident dictionary)
6 R->BASE ;S
7
8
9
10
11
12
13
14
15

```

```

BLOCK #18 ( old TIF #44 )
0 ( TRACE COLON WORDS-FORTH DIMENSIONS III/2 P.58 260CT82 LCT)
1 0 CLOAD (TRACE)      CR ." loading colon definition tracing "
2 FORTH DEFINITIONS
3 0 VARIABLE TRACF    ( CONTROLS INSERTION OF TRACE ROUTINE )
4 0 VARIABLE TFLAG    ( CONTROLS TRACE OUTPUT )
5 : TRACE 1 TRACF ! ;
6 : UNTRACE 0 TRACF ! ;
7 : TRON 1 TFLAG ! ;
8 : TROFF 0 TFLAG ! ;
9 : (TRACE) TFLAG @      ( GIVE TRACE OUTPUT? )
10 IF CR R 2- NFA ID.    ( BACK TO PFA NFA FOR NAME )
11   .S ENDIF ;          ( PRINT STACK CONTENTS )
12 : : ( REDEFINED TO INSERT TRACE WORD AFTER COLON )
13 ?EXEC !CSP CURRENT @ CONTEXT ! CREATE [ ' : CFA @ ] LITERAL
14 HERE 2- ! TRACF @ IF ' (TRACE) CFA DUP @ HERE 2- ! , ENDIF ]
15 ; IMMEDIATE

```

```

BLOCK #19 ( old TIF #72)
0 ( ALTERNATE I/O SUPPORT FOR RS232 PNTR 12JUL82 LCT...mod LES)
1 0 CLOAD INDEX        CR ." loading printing routines"
2 0 0 0 FILE >RS232   BASE->R HEX
3 : SWCH >RS232 PABS @ 10 + DUP PAB-ADDR ! 1- PAB-VBUF !
4   SET-PAB OUTPT F-D" RS232.BA=9600"          OPN 3
5   PAB-ADDR @ VSBW 1 PAB-ADDR @ 5 + VSBW PAB-ADDR @ ALTOUT ! ;
6 : UNSWCH 0 ALTOUT ! CLSE ;
7 : ?ASCII ( BLOCK# --- FLAG )
8   BLOCK 0 SWAP DUP 400 + SWAP
9   DO I C@ 20 > + I C@ DUP 20 < SWAP 7F > OR
10  IF DROP 0 LEAVE ENDIF LOOP ;
11 : TRIAD 0 SWAP SWCH 3 / 3 * 1+ DUP 3 + SWAP
12 DO I ?ASCII IF 1+ I LIST CR ENDIF LOOP
13 -DUP IF 3 SWAP - 14 * 0 DO CR LOOP
14 ." fbForth --- a TI-Forth/fig-Forth extension" 0C EMIT
15 ENDIF UNSWCH ;          R->BASE -->

```

```

BLOCK #20 ( old TIF #73)
0 ( SMART TRIADS AND INDEX 15SEP82 LA0...mod LES )
1 BASE->R DECIMAL      ." ."
2 : TRIADS ( from to --- )
3   3 / 3 * 2+ SWAP 3 / 3 * 1+ DO I TRIAD 3 +LOOP ;
4 : INDEX ( from to --- ) 1+ SWAP
5 DO I DUP ?ASCII IF CR 4 .R 2 SPACES I BLOCK 64 TYPE ELSE DROP
6   ENDIF PAUSE IF LEAVE ENDIF LOOP ; R->BASE ;S
7
8
9
10
11
12
13
14
15

```

## BLOCK #21 ( old TIF #75)

```

0 ( ASSEMBLER 12JUL82 LCT-LES12DEC2013) 0 CLOAD A$$M BASE->R HEX
1 ASSEMBLER DEFINITIONS CR ." loading TMS9900 assembler" CR ." "
2 : GOP' OVER DUP 1F > SWAP 30 < AND IF + , , ELSE + , ENDIF ;
3 : GOP <BUILDS , DOES> @ GOP' ;
4 0440 GOP B,      0680 GOP BL,      0400 GOP BLWP,
5 04C0 GOP CLR,    0700 GOP SET0,    0540 GOP INV,
6 0500 GOP NEG,    0740 GOP ABS,      06C0 GOP SWPB,
7 0580 GOP INC,    05C0 GOP INCT,    0600 GOP DEC,
8 0640 GOP DECT,   0480 GOP X,
9 : GROU <BUILDS , DOES> @ SWAP 40 * + GOP' ;
10 2000 GROU COC,  2400 GROU CZC,  2800 GROU XOR,
11 3800 GROU MPY,  3C00 GROU DIV,  2C00 GROU XOP,
12 : GGOP <BUILDS , DOES> @ SWAP DUP DUP 1F > SWAP 30 < AND
13     IF 40 * + SWAP >R GOP' R> , ELSE 40 * + GOP' ENDIF ;
14 A000 GGOP A,   B000 GGOP AB,   8000 GGOP C,   9000 GGOP CB,
15 6000 GGOP S,   7000 GGOP SB,   E000 GGOP SOC,  F000 GGOP SOCB, -->

```

## BLOCK #22 ( old TIF #76)

```

0 ( ASSEMBLER 12JUL82 LCT) ." ."
1 4000 GGOP SZC,  5000 GGOP SZCB,  C000 GGOP MOV,  D000 GGOP MOVB,
2 : 0OP <BUILDS , DOES> @ , ;
3 0340 0OP IDLE,  0360 0OP RSET,  03C0 0OP CKOF,
4 03A0 0OP CKON,  03E0 0OP LREX,  0380 0OP RTWP,
5 : ROP <BUILDS , DOES> @ + , ; 02C0 ROP STST,  02A0 ROP STWP,
6 : IOP <BUILDS , DOES> @ , , ; 02E0 IOP LWPI,  0300 IOP LIM1,
7 : RIOP <BUILDS , DOES> @ ROT + , , ; 0220 RIOP AI,
8 0240 RIOP ANDI,  0280 RIOP CI,  0200 RIOP LI,  0260 RIOP ORI,
9 : RCOP <BUILDS , DOES> @ SWAP 10 * + + , ;
10 0A00 RCOP SLA,  0800 RCOP SRA,  0B00 RCOP SRC,  0900 RCOP SRL,
11 : DOP <BUILDS , DOES> @ SWAP 00FF AND OR , ;
12 1300 DOP JEQ,  1500 DOP JGT,  1B00 DOP JH,  1400 DOP JHE,
13 1A00 DOP JL,  1200 DOP JLE,  1100 DOP JLT,  1000 DOP JMP,
14 1700 DOP JNC,  1600 DOP JNE,  1900 DOP JNO,  1800 DOP JOC,
15 1C00 DOP JOP,  1D00 DOP SBO,  1E00 DOP SBZ,  1F00 DOP TB, -->

```

## BLOCK #23 ( old TIF #77)

```

0 ( ASSEMBLER 12JUL82 LCT) ." ." CR ." "
1 : GCOP <BUILDS , DOES> @ SWAP 000F AND 040 * + GOP' ;
2 3000 GCOP LDCR,  3400 GCOP STCR,
3 00 CONSTANT R0  01 CONSTANT R1  02 CONSTANT R2  03 CONSTANT R3
4 04 CONSTANT R4  05 CONSTANT R5  06 CONSTANT R6  07 CONSTANT R7
5 08 CONSTANT R8  09 CONSTANT R9  0A CONSTANT R10  0B CONSTANT R11
6 0C CONSTANT R12  0D CONSTANT R13  0E CONSTANT R14
7 0F CONSTANT R15  08 CONSTANT UP  09 CONSTANT SP  0A CONSTANT W
8 0D CONSTANT IP  0E CONSTANT RP  0F CONSTANT NEXT
9 : @() 020 ; : *? 010 + ; : *?+ 030 + ; : @(?) 020 + ;
10 : @(R0) R0 @(?) ; : *R0 R0 *? ; : *R0+ R0 *?+ ;
11 : @(R1) R1 @(?) ; : *R1 R1 *? ; : *R1+ R1 *?+ ;
12 : @(R2) R2 @(?) ; : *R2 R2 *? ; : *R2+ R2 *?+ ;
13 : @(R3) R3 @(?) ; : *R3 R3 *? ; : *R3+ R3 *?+ ;
14 : @(R4) R4 @(?) ; : *R4 R4 *? ; : *R4+ R4 *?+ ;
15 : @(R5) R5 @(?) ; : *R5 R5 *? ; : *R5+ R5 *?+ ; -->

```

```

BLOCK #24 ( old TIF #78)
0 ( ASSEMBLER 12JUL82 LCT)                ." ."
1 : @(R6) R6 @(?); : *R6 R6 *? ; : *R6+ R6 *?+ ;
2 : @(R7) R7 @(?); : *R7 R7 *? ; : *R7+ R7 *?+ ;
3 : @(R8) R8 @(?); : *R8 R8 *? ; : *R8+ R8 *?+ ;
4 : @(R9) R9 @(?); : *R9 R9 *? ; : *R9+ R9 *?+ ;
5 : @(R10) R10 @(?); : *R10 R10 *? ; : *R10+ R10 *?+ ;
6 : @(R11) R11 @(?); : *R11 R11 *? ; : *R11+ R11 *?+ ;
7 : @(R12) R12 @(?); : *R12 R12 *? ; : *R12+ R12 *?+ ;
8 : @(R13) R13 @(?); : *R13 R13 *? ; : *R13+ R13 *?+ ;
9 : @(R14) R14 @(?); : *R14 R14 *? ; : *R14+ R14 *?+ ;
10 : @(R15) R15 @(?); : *R15 R15 *? ; : *R15+ R15 *?+ ;
11 : @(UP) UP @(?); : *UP UP *? ; : *UP+ UP *?+ ;
12 : @(SP) SP @(?); : *SP SP *? ; : *SP+ SP *?+ ;
13 : @(W) W @(?); : *W W *? ; : *W+ W *?+ ;
14 : @(IP) IP @(?); : *IP IP *? ; : *IP+ IP *?+ ;
15 -->

BLOCK #25 ( old TIF #79)
0 ( ASSEMBLER 12JUL82 LCT)                ." ."
1 : @(RP) RP @(?); : *RP RP *? ; : *RP+ RP *?+ ;
2 : *NEXT+ NEXT *?+ ; : *NEXT NEXT *? ; : @(NEXT) NEXT @(?);
3 : @@ @() ; : ** *? ; : *+ *?+ ; : () @(?); ( Wycove syntax)
4
5 ( DEFINE JUMP TOKENS )
6 : GTE 1 ; : H 2 ; : NE 3 ; : L 4 ; : LTE 5 ; : EQ 6 ;
7 : OC 7 ; : NC 8 ; : OO 9 ; : HE 0A ; : LE 0B ; : NP 0C ;
8 : LT 0D ; : GT 0E ; : NO 0F ; : OP 10 ;
9 : CJMP ?EXEC
10 CASE LT OF 1101 , 0 ENDOF GT OF 1501 , 0 ENDOF
11 NO OF 1901 , 0 ENDOF OP OF 1C01 , 0 ENDOF
12 DUP 0< OVER 10 > OR IF 19 ERROR ENDF DUP
13 ENDCASE 100 * 1000 + , ;
14 : IF, ?EXEC [COMPILE] CJMP HERE 2- 42 ; IMMEDIATE
15 -->

BLOCK #26 ( old TIF #80)
0 ( ASSEMBLER 12JUL82 LCT)                ." ."
1 : ENDIF, ?EXEC
2 42 ?PAIRS HERE OVER - 2- 2 / SWAP 1+ C! ; IMMEDIATE
3 : ELSE, ?EXEC 42 ?PAIRS 0 [COMPILE] CJMP HERE 2- SWAP 42
4 [COMPILE] ENDIF, 42 ; IMMEDIATE
5 : BEGIN, ?EXEC HERE 41 ; IMMEDIATE
6 : UNTIL, ?EXEC SWAP 41 ?PAIRS [COMPILE] CJMP HERE - 2 / 00FF
7 AND HERE 1- C! ; IMMEDIATE
8 : AGAIN, ?EXEC 0 [COMPILE] UNTIL, ; IMMEDIATE
9 : REPEAT, ?EXEC >R >R [COMPILE] AGAIN,
10 R> R> 2- [COMPILE] ENDIF, ; IMMEDIATE
11 : WHILE, ?EXEC [COMPILE] IF, 2+ ; IMMEDIATE
12 ( : NEXT, *NEXT B, ; ) ( <--now in kernel )
13 : RT, R11 ** B, ; ( RT pseudo-instruction )
14 : THEN, [COMPILE] ENDIF, ; IMMEDIATE ( ENDIF, synonym )
15 FORTH DEFINITIONS : A$M ; R->BASE

```

**BLOCK #27**

```
0 ( TMS9900 Assembler BLOAD)
1 ." loading TMS9900 assembler "
2 BASE->R 28 R->BASE BLOAD
3 : BLERR IF ." BLOAD error!" THEN ; BLERR FORGET BLERR
4 FORTH DEFINITIONS ;S
5
6
7
8
9
10
11
12
13
14
15
```

**BLOCK #28 – BLOCK #31 TMS9900 Assembler Binary**

```
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
```

**BLOCK #32**

```
0 ( 64-Column Editor BLOAD)
1 ." loading 64-column editor "
2 BASE->R 33 R->BASE BLOAD
3 : BLERR IF ." BLOAD error!" THEN ; BLERR FORGET BLERR
4 FORTH DEFINITIONS ;S
5
6
7
8
9
10
11
12
13
14
15
```

**BLOCK #33 – BLOCK #35 64-Column Editor Binary**

```

0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

```

**BLOCK #36**

```

0 ( TurboForth [MRW] Disk Catalog Utility..mod 19JUN2015 LES)
1 0 CLOAD DIR 0 CLOAD CAT CR ." loading DIR catalog utility"
2 BASE->R HEX 0 VARIABLE CatRec 24 ALLOT
3 1152 @ CatRec OVER 46 + FILE Cat 0 VARIABLE Total
4 0 VARIABLE FCount 0 VARIABLE LC 0 VARIABLE bpr
5 0 VARIABLE sect 0 VARIABLE prot 0B10 VARIABLE Tabs 1C00 ,
6 : Tab ( n --- ) Tabs + C@ CURPOS @ SCRN_WIDTH @ / GOTOXY ;
7 : @R100 9 * CatRec DUP C@ + 2+ + PAD 8 CMOVE PAD F@ F->S ;
8 : DskInfo RD DROP CR CatRec COUNT ." Disk Name: " TYPE CR
9 ." Total: " 1 @R100 DUP U.
10 ." Free: " 2 @R100 DUP U. ." Used: " - U. CR ;
11 : Ftype ( ftype --- ) 2 @R100 bpr ! CASE 1 OF ." DIS/FIX"
12 ENDOF 2 OF ." DIS/VAR" ENDOF 3 OF ." INT/FIX" ENDOF 4 OF
13 ." INT/VAR" ENDOF 5 OF ." PROGRAM" 0 bpr ! ENDOF
14 ." ????????" 0 bpr ! ENDCASE
15 bpr @ -DUP IF 4 U.R THEN ; R->BASE -->

```

**BLOCK #37**

```

0 BASE->R DECIMAL ." ."
1 : Head1 ( --- ) ." ----- ---- ---- -" CR ;
2 : Head ( --- ) ." Name Size Type B/R P" CR Head1 ;
3 : DoDIR 0 LC ! 0 Total ! 0 FCount ! Head BEGIN
4 LC @ 20 MOD 19 = IF KEY DROP CR Head THEN RD DROP
5 CatRec COUNT DUP WHILE TYPE
6 1 @R100 DUP 1- sect ! DUP 0 Tab 4 U.R Total +! 0 @R100 DUP
7 prot ! ABS 1 Tab Ftype prot @ 0< IF 2 Tab ." Y" THEN CR
8 1 LC +! 1 FCount +! REPEAT
9 DROP DROP Head1
10 FCount @ . ." files" 0 Tab Total @ 4 U.R ." sectors" CR ;
11 R->BASE -->
12
13
14
15

```

## BLOCK #38

```

0 BASE->R                ." ."
1 : DIR
2   Cat SET-PAB          ( Initialize PAB skeleton)
3   INTRNL FXD RLTV INPT 38 REC-LEN
4   ( Get directory name from input stream)
5   PAB-ADDR @ 10 + 32 WORD HERE COUNT >R SWAP R VMBW R> N-LEN!
6   ( Get the catalog and display it)
7   OPN                  ( open the catalog)
8   DskInfo              ( display disk info)
9   DoDIR                ( display file list)
10  CLSE                 ( close the catalog) ;
11 R->BASE CR
12 ." DIR - Catalogs a disk." CR
13 ." E.g., DIR DSK1." CR
14
15

```

## BLOCK #39

```

0 ( ASM>CODE [port of Mark Wills' code] LES20JUN2016)
1 CR ." Loading ASM>CODE" 0 CLOAD ASM>CODE BASE @ HEX
2 0 VARIABLE pfa 0 VARIABLE STRPOS 0 VARIABLE FBUF 4E ALLOT
3 PABS @ FBUF 1200 FILE FileOut FileOut SET-PAB
4 : ClearBUF FBUF 50 BLANKS ; : SetFileName ( IS:fileName )
5   BL WORD HERE PAB-ADDR @ 9 + OVER C@ 1+ VMBW ;
6 : ApdERR ( 0 msg# -- flag ) DROP PAB-ADDR @ 1+ VSBR 0E0 AND
7   OR R> R> DROP >R ; : instApdERR ' ApdERR CFA ' (ABORT) !
8 -1 WARNING ! ; : uninstApdERR ' ABORT CFA ' (ABORT) ! 1
9 WARNING ! ; : OpenFile ( -- ) FileOut DSPLY VRBL 50 REC-LEN
10 instApdERR 0 APPND OPN uninstApdERR IF OUTPT OPN THEN ;
11 : Asm? pfa @ DUP CFA @ = ;
12 : copyStr ( addr count -- ) STRPOS @ 5 * FBUF + SWAP CMOVE ;
13 : SetName ClearBUF S" CODE: " COUNT copyStr pfa @ NFA DUP C@
14   01F AND SWAP 1+ SWAP FBUF 6 + SWAP 0 DO OVER C@ 07F AND
15   OVER C! 1+ SWAP 1+ SWAP LOOP DROP DROP ; -->

```

## BLOCK #40

```

0 ( ASM>CODE..continued LES20JUN2016) ." ."
1 : FlushLine 40 WRT ClearBUF 0 STRPOS ! ; : PlaceCell pfa @ @
2 0 <# # # # #> copyStr 1 STRPOS +! 2 pfa +! ;
3 : & S" ;CODE" COUNT copyStr ;
4 : ProcessWord SetName FlushLine BASE->R 10 BASE ! BEGIN pfa @ @
5   045F = 0= WHILE PlaceCell STRPOS @ 0C = IF FlushLine THEN
6   REPEAT &; FlushLine R->BASE ;
7 : ASM>CODE ( IS:wordName fileName) CR -FIND IF DROP ELSE 0 THEN
8   pfa ! SetFileName pfa @ IF Asm? IF OpenFile ProcessWord CLSE
9   ELSE ." Not an assembly language word" THEN
10  ELSE ." Word not found" THEN ;
11 CR ." Usage: ASM>CODE <name> <file>"
12 CR ." E.g.: ASM>CODE MYWORD DSK1.MYWORD" CR BASE ! ;S
13
14
15

```

## BLOCK #41

```

0 ( Useful words--most are required by fbForth String Library)
1 ( written by Mark Wills, Lee Stewart & Marshall Linker)
2 0 CLOAD $.      CR ." Loading useful additional words--" CR
3 ." 2DUP 2DROP NIP TUCK CELLS -ROT PICK ROLL WITHIN <> $. EXIT"
4 : 2DUP ( a b -- a b a b ) OVER OVER ;
5 : 2DROP ( a b -- ) DROP DROP ; : NIP ( a b -- b ) SWAP DROP ;
6 : TUCK ( a b -- b a b ) SWAP OVER ; : CELLS ( n -- 2n ) 2 * ;
7 : -ROT ( a b c -- c a b ) ROT ROT ;
8 : PICK ( +n -- [n] ) 1+ CELLS SP@ + @ ;
9 ( The source for ROLL was Marshall Linker via
10  George Smyth's Forth Forum)
11 : ROLL ( [n]..[0] +n -- [n-1]..[0][n] )
12   -DUP IF 1- SWAP >R MYSELF R> SWAP THEN ;
13 : WITHIN ( n low high -- true|false ) OVER - >R - R> U< ;
14 : <> ( a b -- 1|0 ) = 0= ; : $. BASE->R HEX U. R->BASE ;
15 : EXIT ( -- ) [COMPILE] ;S ; IMMEDIATE

```

## BLOCK #42

```

0 ( Portable, Stack Based String Library for fbForth V2.0 )
1 ( V 1.0 - Mark Wills Sept 2014.)
2 ( Ported from the original TurboForth code by Mark Wills )
3 ( Modified by Lee Stewart October 2014)
4 BASE->R DECIMAL 41 R->BASE CLOAD $.
5 CR ." Loading String Library"
6 0 CONSTANT ($sSize)
7 HERE CONSTANT ($sEnd)
8 ($sEnd) VARIABLE ($sp)
9 0 VARIABLE ($temp1)
10 0 VARIABLE ($depth)
11 0 VARIABLE ($temp0)
12 0 VARIABLE ($temp2)
13 0 VARIABLE ($temp3)      -->
14
15

```

## BLOCK #43

```

0 ( Throw codes for string library, mod: Lee Stewart)
1 BASE->R DECIMAL      ." ."
2 : (throw) ( code -- )
3   CASE
4     ($sSize) 0= IF DROP 9999 THEN
5       9900 OF ." String stack underflow" ENDOF
6       9901 OF ." String too large to assign" ENDOF
7       9902 OF ." String stack is empty" ENDOF
8       9903 OF ." Need at least 2 strings on string stack" ENDOF
9       9904 OF ." String too large for string constant" ENDOF
10      9905 OF ." Illegal LEN value" ENDOF
11      9906 OF ." Need at least 3 strings on string stack" ENDOF
12      9908 OF ." Illegal start value" ENDOF
13      9999 OF ." String stack not initialized" ENDOF
14    ENDCASE
15    CR ABORT ;      R->BASE  -->

```

## BLOCK #44

```

0 ( String stack words, mod: Lee Stewart [INIT$ added])
1 : ($depth+) ( -- ) 1 ($depth) +! ;      BASE->R DECIMAL ." ."
2 : ($sp@) ( -- addr ) ($sp) @ ;
3 : ($rUp) ( n -- n|n+1 ) 1+ -2 AND ;
4 : cell+ ( n -- n+2 ) COMPILE 2+ ; IMMEDIATE
5 : (sizeOf$) ( $addr - $size ) @ ($rUp) cell+ ;
6 : (set$SP) ( $size -- ) MINUS DUP ($sp@) + ($sEnd)
7   < IF 9900 (throw) THEN ($sp) +! ;
8 : (addrOf$) ( index -- addr ) ($sp@) SWAP DUP IF 0 DO
9   DUP (sizeOf$) + LOOP ELSE DROP THEN ;
10 : (lenOf$) ( $addr -- len )
11   STATE @ IF COMPILE @ ELSE @ THEN ; IMMEDIATE
12 : INIT$ ( stack_size -- ) ' ($sSize) ! HERE ' ($sEnd) !
13   ($sEnd) ($sSize) + ($sp) ! ($sSize) ALLLOT ;
14 : RESET$ ( -- ) 0 ($depth) ! ($sEnd) ($sSize) + ($sp) ! ;
15 : DEPTH$ ( -- $sDepth) ($depth) @ ;      R->BASE -->

```

## BLOCK #45

```

0 ( String constant words etc.)      BASE->R DECIMAL ." ."
1 : $CONST ( max_len tib:"name" -- ) ( runtime: -- $Caddr)
2   <BUILDS ($rUp) DUP , 0 , ALLLOT DOES> NOP ;
3 : CLEN$ ( $Caddr -- len ) cell+ @ ;
4 : MAXLEN$ ( $Caddr -- max_len ) (lenOf$) ;
5 : .$CONST ( $Caddr -- ) cell+ DUP (lenOf$)
6   SWAP cell+ SWAP TYPE ;
7 : := ( $Caddr tib:"string" -- ) DUP @ 34 WORD HERE COUNT
8   SWAP >R 2DUP < IF 9901 (throw) THEN NIP 2DUP SWAP cell+
9   ! >R [ 2 CELLS ] LITERAL + R> R> -ROT CMOVE ;
10 : ($") ( addr len -- ) ( ss: -- str ) DUP ($rUp) cell+ (set$SP)
11   DUP ($sp@) ! ($sp@) cell+ SWAP CMOVE ($depth+) ;
12 : (COMPILE$) ( addr len -- ) DUP >R PAD SWAP CMOVE HERE 6 CELLS
13   COMPILE LIT + , COMPILE LIT R , COMPILE BRANCH HERE R
14   ($rUp) + HERE - 2+ , PAD 12 - R HERE SWAP CMOVE R> ($rUp)
15   ALLLOT COMPILE ($") ;      R->BASE -->

```

## BLOCK #46

```

0 ( String stack words)      BASE->R DECIMAL ." ."
1 : $" 34 WORD HERE COUNT STATE @ IF (COMPILE$) ELSE ($") THEN ;
2   IMMEDIATE : >$ cell+ DUP (lenOf$) SWAP cell+ SWAP ($") ;
3 : PICK$ DEPTH$ 0= IF 9902 (throw) THEN
4   (addrOf$) DUP (lenOf$) SWAP cell+ SWAP ($") ;
5 : DUP$ DEPTH$ 0= IF 9902 (throw) THEN 0 PICK$ ;
6 : DROP$ DEPTH$ 0= IF 9902 (throw) THEN
7   ($sp@) (sizeOf$) MINUS (set$SP) -1 ($depth) +! ;
8 : SWAP$ DEPTH$ 2 < IF 9903 (throw) THEN ($sp@) DUP (sizeOf$)
9   HERE SWAP CMOVE 1 (addrOf$) DUP (sizeOf$) ($sp@) SWAP CMOVE
10  HERE DUP (sizeOf$) ($sp@) DUP (sizeOf$) + SWAP CMOVE ;
11 : NIP$ DEPTH$ 2 < IF 9903 (throw) THEN SWAP$ DROP$ ;
12 : OVER$ DEPTH$ 2 < IF 9903 (throw) THEN 1 PICK$ ;
13 : (rot$) ($sp@) 3 (addrOf$) ($sp@) (sizeOf$)
14   1 (addrOf$) (sizeOf$) 2 (addrOf$) (sizeOf$) + + CMOVE
15   3 (addrOf$) ($sp) ! -3 ($depth) +! ;      R->BASE -->

```

## BLOCK #47

```

0 ( String stack words)          BASE->R DECIMAL      ." ."
1 : ROT$   DEPTH$ 3 < IF 9906 (throw) THEN
2   1 PICK$ 1 PICK$ 4 PICK$ (rot$) ;
3 : -ROT$   DEPTH$ 3 < IF 9906 (throw) THEN
4   0 PICK$ 3 PICK$ 3 PICK$ (rot$) ;
5 : LEN$   DEPTH$ 1 < IF 9902 (throw) THEN ($sp@) @ ;
6 : >$CONST >R DEPTH$ 1 < IF 9902 (throw) THEN LEN$ R @ > IF 9904
7   (throw) THEN ($sp@) DUP (sizeOf$) R> cell+ SWAP CMOVE DROP$ ;
8 : +$ DEPTH$ 2 < IF 9903 (throw) THEN 1 (addrOf$) cell+ HERE 1
9   (addrOf$) (lenOf$) CMOVE ($sp@) cell+ 1 (addrOf$) (lenOf$) HERE
10 + LEN$ CMOVE HERE LEN$ 1 (addrOf$) (lenOf$) + DROP$ DROP$ ($) ;
11 : MID$   DEPTH$ 1 < IF 9902 (throw) THEN DUP LEN$ > OVER 1 < OR
12   IF 9905 (throw) THEN OVER DUP LEN$ > SWAP 0< OR IF 9908
13   (throw) THEN SWAP ($sp@) cell+ + SWAP ($) ;
14 : LEFT$  DEPTH$ 1 < IF 9902 (throw) THEN DUP LEN$ > OVER 1 < OR
15 IF 9905 (throw) THEN 0 ($sp@) cell+ + SWAP ($) ; R->BASE -->

```

## BLOCK #48

```

0 ( String stack words)          BASE->R DECIMAL      ." ."
1 : RIGHT$ DEPTH$ 1 < IF 9902 (throw) THEN DUP LEN$ > OVER 1 <
2   OR IF 9905 (throw) THEN ($sp@) (lenOf$) OVER -
3   ($sp@) cell+ + SWAP ($) ;
4 : FINDC$ DEPTH$ 1 < IF 9902 (throw) THEN -1 ($temp0) ! ($sp@)
5   cell+ ($sp@) (lenOf$) 0 DO DUP C@ 2 PICK = IF I ($temp0) !
6   LEAVE THEN 1+ LOOP DROP DROP ($temp0) @ ;
7 : FIND$   DEPTH$ 2 < IF 9903 (throw) THEN LEN$ ($temp1) ! 1
8   (addrOf$) (lenOf$) ($temp0) ! DUP ($temp0) @ > IF DROP -1 EXIT
9   THEN 1 (addrOf$) cell+ + ($temp2) ! ($sp@) cell+ ($temp3) !
10  ($temp1) @ ($temp0) @ > IF DROP -1 EXIT THEN 0 ($temp0) @ 0 DO
11  ($temp3) @ OVER + C@ ($temp2) @ I + C@ = IF 1+ DUP ($temp1) @
12  = IF DROP I ($temp1) @ - 1+ -2 LEAVE THEN ELSE DROP 0 THEN
13  LOOP DUP -2 = IF DROP ELSE DROP -1 THEN DROP$ ;
14 : .$ DEPTH$ 0= IF 9902 (throw) THEN
15   ($sp@) cell+ ($sp@) (lenOf$) TYPE DROP$ ; R->BASE -->

```

## BLOCK #49

```

0 ( String stack words)          BASE->R DECIMAL      ." ."
1 : REV$   DEPTH$ 0= IF 9902 (throw) THEN ($sp@) DUP cell+ >R
2   (lenOf$) R> SWAP HERE SWAP CMOVE ($sp@) (lenOf$) HERE 1- +
3   ($sp@) cell+ DUP ($sp@) (lenOf$) + SWAP DO
4   DUP C@ I C! 1- LOOP DROP ;
5 : LTRIM$ DEPTH$ 0= IF 9902 (throw) THEN ($sp@) DUP (lenOf$) >R
6   HERE OVER (sizeOf$) CMOVE 0 R> HERE cell+ DUP >R + R> DO I C@
7   BL = IF 1+ ELSE LEAVE THEN LOOP DUP 0 > IF >R ($sp@) (lenOf$)
8   DROP$ HERE cell+ R + SWAP R> - ($) ELSE DROP THEN ;
9 : RTRIM$ DEPTH$ 0= IF 9902 (throw) THEN REV$ LTRIM$ REV$ ;
10 : UCASE$ DEPTH$ 1 < IF 9902 (throw) THEN ($sp@) DUP (lenOf$) +
11   cell+ ($sp@) cell+ DO I C@ DUP 97 123 WITHIN IF 32 - I
12   C! ELSE DROP THEN LOOP ; : TRIM$ RTRIM$ LTRIM$ ;
13 : LCASE$ DEPTH$ 1 < IF 9902 (throw) THEN ($sp@) DUP (lenOf$) +
14   cell+ ($sp@) cell+ DO I C@ DUP 65 91 WITHIN IF
15   32 + I C! ELSE DROP THEN LOOP ; R->BASE -->

```

## BLOCK #50

```

0 ( String stack words, mod: LES [CMP$ added])
1 BASE->R DECIMAL      ." ."
2 : REPLACE$ DEPTH$ 3 < IF 9906 (throw) THEN LEN$ >R 0 FIND$ DUP
3   ($temp0) ! -1 > IF ($sp@) cell+ HERE ($temp0) @ CMOVE 1
4   (addrOf$) cell+ HERE ($temp0) @ + 1 (addrOf$) (lenOf$) CMOVE
5   ($sp@) cell+ ($temp0) @ + R + HERE ($temp0) @ + 1 (addrOf$)
6   (lenOf$) + LEN$ R> - ($temp0) @ - DUP >R CMOVE R> ($temp0)
7   @ + 1 (addrOf$) (lenOf$) + DROP$ DROP$ HERE SWAP ($)
8   ELSE R> DROP THEN ($temp0) @ ;
9 : CMP$ DEPTH$ 2 < IF 9903 (throw) THEN 1 (addrOf$) cell+ ($sp@)
10  cell+ 1 (addrOf$) (lenOf$) LEN$ OVER OVER - SGN >R MIN 0
11  SWAP 0 DO DROP OVER I + C@ OVER I + C@ - SGN DUP IF LEAVE
12  THEN LOOP R> OVER 0= IF OR ELSE DROP THEN -ROT DROP DROP ;
13 : VAL$ ($sp@) DUP (lenOf$) >R cell+ PAD 1+ R CMOVE
14   R PAD C! 32 PAD R> + 1+ C! PAD NUMBER DROP$ ;
15                                     R->BASE  -->

```

## BLOCK #51

```

0 ( String stack words)      BASE->R DECIMAL      ." ." CR
1 : $.S CR DEPTH$ 0 > IF ($sp@) DEPTH$ ." Index|Length|String"
2   CR ." -----+-----+-----" CR 0 BEGIN DEPTH$ 0 > WHILE DUP
3   6 .R ." |" LEN$ 6 .R ." |" $. 1+ CR REPEAT DROP ($depth) !
4   ($sp) ! CR ELSE ." String stack is empty." CR THEN
5   ." Allocated stack space:" ($sEnd) ($sSize) + ($sp@) - 4 .R
6   ." bytes" CR ." Total stack space:" ($sSize) 4 .R
7   ." bytes" CR ." Stack space remaining:" ($sp@) ($sEnd) - 4
8   .R ." bytes" CR ;      R->BASE
9 ." You MUST initialize the string stack before you can use the
10 string library:" CR
11 ." 512 INIT$" CR
12 ." will create a string stack with 512 bytes available." CR
13 ." Example: $" 34 EMIT ." RED" 34 EMIT ." $" 34 EMIT
14 ." GREEN" 34 EMIT ." $" 34 EMIT ." BLUE" 34 EMIT ." $.S"
15 CR

```

## BLOCK #52

```

0 ( String Library BLOAD)
1 ." loading string library " CR
2 ." You MUST initialize the string stack before you can use the
3 string library:" CR
4 ." 512 INIT$" CR
5 ." will create a string stack with 512 bytes available." CR
6 ." Example: $" 34 EMIT ." RED" 34 EMIT ." $" 34 EMIT
7 ." GREEN" 34 EMIT ." $" 34 EMIT ." BLUE" 34 EMIT ." $.S"
8 CR
9 BASE->R DECIMAL 53 R->BASE BLOAD
10 : BLERR IF ." BLOAD error!" THEN ; BLERR FORGET BLERR
11 FORTH DEFINITIONS ;S
12
13
14
15

```

**BLOCK #53 – BLOCK #56 String Library Binary**

0  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15

**BLOCK #57**

0  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15

**BLOCK #58**

```
0 ( Catalog program that uses VIB, FDIR and FDRs..LES 11NOV2015)
1 0 CLOAD CAT 0 CLOAD DIR
2 BASE->R CR          ." loading CAT catalog program"
3 HEX 0 VARIABLE Buf1 0FE ALLOT 0 VARIABLE Buf2 12 ALLOT
4 0 VARIABLE Total 0 VARIABLE FCount 0 VARIABLE LC
5 0 VARIABLE bpr 0 VARIABLE sect 0 VARIABLE prot
6 1154 CONSTANT VBuf 0B10 VARIABLE Tabs 181E ,
7 0110 VARIABLE CATPAB
8 : RdErr? ( err --- ) -DUP IF CR ." Disk I/O error "
9  BASE->R [COMPILE] HEX . R->BASE ABORT THEN ;
10 : DSRLNK10 0A 0E SYSTEM 8350 C@ RdErr? ;
11 : getBuf ( bufadr count --- ) VBuf ROT ROT VMBR ;
12 : getSect ( sect# --- ) 8350 ! VBuf 834E ! VBuf 2- 8356 !
13 DSRLNK10 ;
14 : Tab ( n --- ) Tabs + C@ CURPOS @ SCRN_WIDTH @ / GOTOXY ;
15 R->BASE -->
```

## BLOCK #59

```

0 BASE->R DECIMAL                ." ."
1 : getFree ( --- n ) 0 Buf2 ! Buf1 56 + DUP 200 + SWAP DO
2 I @ 65535 XOR -DUP IF 16 0 DO DUP 1 AND Buf2 +! 1 SRL LOOP
3 DROP THEN 2 +LOOP Buf2 @ ; : Head1 ( --- )
4 ." ----- -" CR ; : Head ( --- )
5 ." Name Size Typ B/R Bytes P" CR Head1 ; HEX
6 : DskInfo ( dsk# --- ) SWPB 1+ 834C ! 0 getSect Buf1 100 getBuf
7 CR Buf1 0A ." Disk Name: " TYPE CR ." Total: " Buf1 0A + @ 2-
8 DUP U. ." Free: " getFree DUP U.
9 ." Used: " - U. CR ; DECIMAL
10 : Ftype ( --- ) Buf2 17 + C@ bpr ! Buf2 12 + C@ DUP 8 AND
11 prot ! 247 AND CASE 0 OF ." D/F" ENDOF 128 OF
12 ." D/V" ENDOF 2 OF ." I/F" ENDOF 130 OF ." I/V"
13 ENDOF 1 OF ." PGM" sect @ 256 * Buf2 16 + C@ -DUP IF +
14 256 - THEN 0 bpr ! 2 Tab 5 U.R ENDOF ." ????" 0 bpr !
15 ENDCASE bpr @ -DUP IF 4 U.R THEN ; R->BASE -->

```

## BLOCK #60

```

0 BASE->R DECIMAL                ." ."
1 : DoCAT ( --- ) 0 LC ! 0 Total ! 0 FCount ! Head 1 getSect
2 Buf1 256 getBuf Buf1 BEGIN LC @ 20 MOD 19 = IF KEY DROP
3 CR Head THEN DUP @ -DUP WHILE getSect Buf2 20 getBuf Buf2 10
4 TYPE Buf2 14 + @ DUP sect ! 1+ DUP 0 Tab 4 U.R Total +! 1 Tab
5 Ftype prot @ IF 3 Tab ." Y" THEN CR 1 LC +! 1 FCount +! 2+
6 REPEAT DROP Head1 FCount @ . ." files" 0 Tab Total @ 4 U.R
7 ." sectors" CR ;
8 : CAT ( dsk# --- ) BASE->R [COMPILE] DECIMAL
9 CATPAB VBuf 2- 2 VMBW DskInfo DoCAT R->BASE ;
10 CR ." n CAT - Catalogs a disk. n = disk #." CR
11 ." E.g., 1 CAT catalogs DSK1." CR R->BASE ;S
12
13
14
15

```

## BLOCK #61

```

0 ( TI Forth disk browser/copier..LES 04DEC2015) BASE->R HEX
1 CR ." loading TI Forth Viewer/Copier"
2 1154 CONSTANT VTibuf 0110 VARIABLE TIPAB 1 VARIABLE Dsk
3 0 VARIABLE outBFL 10 ALLOT 0 VARIABLE curBFL 10 ALLOT
4 : GNUM BL WORD HERE NUMBER DROP ; : getD0idx ( -- lim idx )
5 GNUM GNUM OVER OVER > IF SWAP THEN 1+ SWAP ; : BlkBuf PREV @
6 2+ ; : getDsk ( IS:DSKn) BL WORD HERE 4 + C@ 30 - Dsk ! ;
7 : RdErr? ( err -- ) -DUP IF CR ." Disk I/O error " BASE->R
8 [COMPILE] HEX . R->BASE ABORT THEN ; : DSRLNK10 0A 0E SYSTEM
9 8350 C@ RdErr? ; : getTIblock FLUSH TIPAB VTibuf 2- 2 VMBW
10 VTibuf 834E ! Dsk @ SWPB 1+ 834C ! 2 SLA BlkBuf DUP 400 + SWAP
11 DO DUP 8350 ! 1+ VTibuf 2- 8356 ! DSRLNK10 VTibuf I 100 VMBR
12 100 +LOOP DROP ; : dnLeft CURPOS @ SCRNS_WIDTH @ MOD IF CR THEN
13 ; : EMITG ( n -- ) CURPOS @ VSBW CURPOS @ 1+ DUP SCRNS_END @ <
14 IF CURPOS ! ELSE DROP CR THEN ; : TYPEG ( addr cnt -- ) -DUP
15 IF OVER + SWAP DO I C@ EMITG LOOP ELSE DROP THEN ; R->BASE -->

```

## BLOCK #62

```

0 ( TI Forth disk browser/copier..continued)  BASE->R HEX ." ."
1 : dspLine ( line# -- ) 40 * BlkBuf + 40 TYPEG ;
2 : 64page? CURPOS @ 40 + SCRN_END @ > IF KEY DROP PAGE THEN ;
3 : TIFBLK ( IS:blk# DSKn ) GNUM getDsk getTIBlock PAGE 10 0 DO
4 64page? dnLeft I 2 .R ." | " I dspLine PAUSE IF LEAVE THEN
5 LOOP ; : TIFIDX ( IS:startblk endblk DSKn) getD0idx getDsk
6 PAGE DO I getTIBlock 64page? dnLeft I 3 .R ." | " 0 dspLine
7 PAUSE IF LEAVE THEN LOOP CR ." ...done" ; : gBFL ( -- ) BL
8 WORD HERE outBFL HERE C@ 1+ CMOVE ; : saveCurBFL BPB BPOFF @
9 + 9 + DUP VSBR curBFL SWAP 1+ VMBR ; : getBFL TIB @ 0F EXPECT
10 0 IN ! gBFL ; : cpyTI2FB ( dstBlk# lim idx -- ) CURPOS @ >R
11 DO J CURPOS ! I 3 .R I getTIBlock DUP PREV @ ! UPDATE FLUSH
12 1+ LOOP DROP R> DROP ;
13 : TIF2FBF ( IS:srcStrtBlk srcEndBlk DSKn dstStrtBlk dstBlksFil)
14 saveCurBFL getD0idx getDsk GNUM gBFL outBFL (UB) ROT ROT
15 cpyTI2FB curBFL (UB) ; R->BASE -->

```

## BLOCK #63

```

0 ( TI Forth disk browser/copier..continued)  BASE->R HEX ." ."
1 : BOXCHRS DATA[ 0000 003C 3C30 3030 0000 00F0 F030 3030 3030
2 303C 3C00 0000 3030 30F0 F000 0000 0000 00FC FC00 0000 0000
3 00FC FC30 3030 3030 3030 3030 3060 C070 380C 1830 40A0
4 A8B4 5414 0800 40C0 4854 F414 0800 40A0 2854 F414 0800 C020
5 4834 D414 0800 2060 A8F4 3414 0800 E080 6834 D414 0800 4080
6 C8B4 5414 0800 0000 FC00 FC00 FC00 ]DATA C9 DCHAR ;
7 D1CD VARIABLE TLDATA DATA[ CDCD CDCE CDCD CDCD D2CD CDCD CDCE
8 CDCD CDCD D3CD CDCD CDCE CDCD CDCD D4CD CDCD CDCE CDCD CDCD
9 D5CD CDCD CDCE CDCD CDCD D6CD CDCD CDCE CDCD CDCD D7CD CDCD
10 ]DATA DROP DROP
11 0 VARIABLE TIFblk 0 VARIABLE fbFblk 0 CONSTANT OFFSET
12 : WINWID ( -- winwid ) SCRN_WIDTH @ 28 = IF 22 ELSE 40 THEN ;
13 : CORNERS 3 3 1 0C9 HCHAR 3 14 1 0CB HCHAR 4 WINWID + DUP 3 1
14 0CA HCHAR 14 1 0CC HCHAR ; : TOPLN ( -- ) OFFSET TLDATA + 4
15 3 GOTOXY WINWID TYPEG ; R->BASE -->

```

## BLOCK #64

```

0 ( TI Forth disk browser/copier..continued)  BASE->R HEX ." ."
1 : BOTLN 4 14 WINWID 0CD HCHAR ; : SIDELN ( col chr -- ) 4 10
2 ROT VCHAR ; : SIDELNS 3 0CF SIDELN WINWID 4 + 0CF SIDELN ;
3 : RPT ( chr cnt -- ) 0 DO DUP EMITG LOOP DROP ; : drawScrn
4 PAGE 0D8 6 RPT ." TI Forth Block Viewer/Copier" 0D8 6 RPT
5 VDPME @ 0= IF 0D8 28 RPT THEN ." TI Forth:DSK fbForth:" CR
6 ." Block Block" 0 ' OFFSET ! CORNERS TOPLN BOTLN
7 SIDELNS SCRN_WIDTH @ DUP 4 * BASE->R DECIMAL 10 0 DO DUP
8 CURPOS ! I 3 .R OVER + LOOP R->BASE DROP DROP CR CR
9 ." F4:+Block F6:-Block FD:+Panel FS:-Panel "
10 ." FT:TI# FF:fb# ^F:BlkFil ^S:TI>fb F9:Xit" ; : dspLnSeg
11 ( line# -- ) 40 * BlkBuf OFFSET + + WINWID TYPEG ; : dspBlock
12 SCRN_WIDTH @ 28 = IF 3 26 OFFSET CASE 00 OF 0CF 0D0 ENDOF 0F
13 OF 0D0 0D0 ENDOF 1E OF 0D0 0CF ENDOF ELSEOF 0CF 0CF ENDOF
14 ENDCASE ROT SWAP SIDELN SIDELN TOPLN THEN 10 0 DO SCRN_WIDTH @
15 I 4 + * 4 + CURPOS ! I dspLnSeg LOOP ; R->BASE -->

```

## BLOCK #65

```

0 ( TI Forth disk browser/copier..continued)  BASE->R HEX ." ."
1 : calcOff ( -1|0|+1 -- ) DUP IF 0F * OFFSET + DUP 0< IF DROP 1E
2 THEN DUP 1E > IF DROP 0 THEN THEN ' OFFSET ! ; : dspPanel
3 ( +1|-1 -- ) WINWID 22 = IF calcOff dspBlock ELSE DROP THEN ;
4 : getCmd ( -- key ) ?KEY DUP IF BEGIN ?KEY 0= UNTIL THEN ;
5 : dspBlk# ( n col row -- ) GOTOXY 3 .R ; : get# ( -- n ) TIB @
6 3 EXPECT 0 IN ! BL WORD HERE NUMBER DROP ; : getBlk#
7 ( min col row -- n ) ROT >R OVER OVER GOTOXY CURPOS @ DUP 3 20
8 VFILL CURPOS ! get# DUP R < IF DROP R> ELSE R> DROP THEN DUP
9 >R ROT ROT dspBlk# R> ; : nxtTIBlk ( +1|-1 -- ) TIFblk +!
10 TIFblk @ DUP 8 2 dspBlk# getTIBlock 0 calcOff dspBlock ;
11 : clrLstLn 0 17 SCRN_WIDTH @ 20 HCHAR 0 17 GOTOXY ;
12 : keyPrompt ." ..tap key" KEY DROP clrLstLn ; R->BASE -->
13
14
15

```

## BLOCK #66

```

0 ( TI Forth disk browser/copier..continued)  BASE->R HEX ." ."
1 : cmd ( get command key) BEGIN getCmd CASE 02 OF 1 nxtTIBlk 0
2 END OF 0C OF TIFblk @ IF -1 nxtTIBlk THEN 0 END OF 09 OF 1
3 dspPanel 0 END OF 08 OF -1 dspPanel 0 END OF 5D OF 0 8 2 getBlk#
4 DUP TIFblk ! getTIBlock 0 calcOff dspBlock 0 END OF 7B OF 1 18
5 2 getBlk# fbFblk ! 0 END OF 06 OF 18 1 GOTOXY CURPOS @ DUP 10
6 20 VFILL CURPOS ! getBFL outBFL (UB) 0 END OF 13 OF fbFblk @
7 DUP IF outBFL @ DUP IF SWAP TIFblk @ clrLstLn
8 ." How many blocks? " get# OVER + SWAP clrLstLn cpyTI2FB
9 ." done" keyPrompt ELSE SWAP DROP THEN THEN 0= IF clrLstLn
10 ." fbForth block#|file not set!" keyPrompt THEN 0 END OF 0F OF
11 PAGE 1 END OF ELSE OF 0 END OF ENDCASE UNTIL ;
12 : TIFVU ( IS:blk# DSKn) GNUM DUP TIFblk ! getDsk getTIBlock
13 VDPMDE @ 2 < IF saveCurBFL BOXCHRS drawScrn 0C 1 GOTOXY Dsk @
14 . TIFblk @ 8 2 dspBlk# dspBlock cmd curBFL (UB) ELSE CR
15 ." TEXT or TEXT80 modes only!" THEN ; R->BASE -->

```

## BLOCK #67

```

0 ( TI Forth disk browser/copier..continued)  BASE->R HEX ." ."
1 CR CR ." USAGE:"
2 CR ." TIFBLK <block#> DSKn"
3 CR ."   ex: TIFBLK 2 DSK2"
4 CR ." TIFIDX <strtBlock#> <endBlock#> DSKn"
5 CR ."   ex: TIFIDX 9 40 dsk1"
6 CR ." TIF2FBF <srcStrtBlk#> <srcEndBlk#>"
7 CR ."   DSKn <dstStrtBlk#> <dstFile>"
8 CR ."   ex: TIF2FBF 3 6 DSK3 9 DSK1.MYBLOCKS"
9 CR ." TIFVU <block#> DSKn"
10 CR ."   ex: TIFVU 58 DSK2" CR CR R->BASE ;S
11
12
13
14
15

```