Hidden But Useful TFBASIC Functions

for the TFX-11v2 only
Tuesday, July 27, 2004

Since the TFX-11v2 was released, we have received requests from customers about how to perform certain tasks that were either not available in TFBASIC or were not documented. This document explains how to perform these tasks.

Tasks explained in this document

- Load a program into RAM and run the program
- Erase the datafile using serial port commands
- Off-load the datafile using serial port commands
- Burn a program loaded in RAM into flash EEPROM
- Check if the datafile has been off-loaded using serial port commands
- Read the version of TFBASIC using serial port commands
- Flush data to the EEPROM in a TFBASIC program
- Change the value of DFPNT in a TFBASIC program
- Erase datafile pages from within a TFBASIC program
- Prevent datafile corruption after a power failure
- List of additional Assembly Language Subroutines

Load a program into RAM and run the program

Why you may want to do it
You may want to write your own program to load a TFBASIC program to the TFX-11v2. The program will have to have been compiled and saved in binary format. See "Additional notes" below.

How to do it

	Desk-top computer	TFBASIC action
1)	Send Ctrl-L character	Echoes Ctrl-L if ready
2)	Send byte specifying number of data bytes to follow (1 to 255)	no action
3)	Send the block of bytes	Saves program bytes to RAM
4)	If more bytes to send, go to step 2	no action
5)	Send a zero byte to stop transfer	Sends CR/LF then '#'
6)	Send Ctrl-R character	Runs the program

Additional notes
The '#' prompt must be displayed by the TFX-11v2 before this protocol can begin.

This assumes you've used TFTools to create a compiled binary file of the program that will be loaded into the TFX-11v2. With the program loaded in an editor window, select the item "Compiler options" under the Tattletale menu in TFTools. When you get the Compiler Options dialog, select "Create binary file" and hit "OK". Then select "Syntax check" under the Tattletale menu. This will compile the program and store it in a file with a .bin extension.

There is no time-out on this protocol nor is there any error checking.

Erase the datafile using serial port commands

Why you may want to do it
Normally, the parallel port is used to erase the datafile because this is part of the process of loading a new copy of the operating system and loading a program into the EEPROM. If there is no parallel port available, though, you may need to use the serial port protocol listed here. Unlike versions of TFBASIC previous to version 2, this method of erasing the datafile will set the DFERASED value that can be read by a TFBASIC program. Also, TFBASIC version 2 added the capability to check if the datafile has been off-loaded (since any data was stored) by using the Ctrl-B command. See the section titled “Check if the datafile has been off-loaded using serial port commands” for information about doing this check.

How to do it

Host computer action	TFBASIC action
1) Send Ctrl-E	Echoes Ctrl-E
2) Send one's complement of Ctrl-E (FA hex)	Erases datafile
3)	Sets datafile pointer to 0
4)	Sets DFERASED flag true
5)	Returns '0' if successful Returns '1' if an error occurred

Additional notes
The '#' prompt must be displayed by the TFX-11v2 before this protocol can begin.

The sending of the one's complement of Ctrl-E in Step 2 must be done within one second of receiving the echoed Ctrl-E.

The Tattletale sends a '0' (zero) character if the erase succeeds or a '1' (one) character if the erase fails.

To restart the program, once the '#' sign is showing again, just send a Ctrl-R character.

Off-load the datafile using serial port commands

Make sure you understand the consequences
Normally, the parallel port is used to off-load the datafile because it is over 6 times faster than off-loading the datafile serially at 38400 baud. But sometimes, you don't have access to a parallel port.

How to do it

Host computer action	TFBASIC action
1) Send Ctrl-O	Echoes Ctrl-O
2) Send Ctrl-U or Ctrl-T	If a Ctrl-U is received, it is assumed that this is a simple XMODEM transfer and it skips to step 7). If a Ctrl-T is received, it is assumed that the desk-top application is requesting the total size of the datafile and it continues to step 3).
3) (after Ctrl-T)	Sends total size of datafile as 8 hex characters followed by CR/LF. Bytes are sent MSB first.
4) Send Ctrl-N	Sends current datafile pointer as 8 hex characters followed by CR/LF. Sends MSB first.
5) Send number of bytes to off-load as 8 hex char followed by CR/LF. Send MSB first.	
6) Send Ctrl-U	
7) (after Ctrl-U) Start XMODEM protocol	Responds to XMODEM

Additional notes
The '#' prompt must be displayed by the TFX-11v2 before this protocol can begin.

To restart the program, once the '#' sign is showing again, just send a Ctrl-R character.

In versions of TFBASIC previous to version 2.07.37, there were two extra steps before step 6). TFBASIC would send a Ctrl-Z character to query whether Page 0 (the Header page, not containing any data) should be off-loaded before the datafile. The desk-top computer would then respond with either a Ctrl-B to request that Page 0 be included or any other character to decline. Either way, TFBASIC 2.00 and above would not send Page 0. This was included to be compatible with older versions of TFBASIC but was found to be of no use to customers.

When the notes above refer to sending 8 hex characters, it means to send the ASCII byte corresponding to the characters. For instance, to send the value 2079232 (a typical value for the size of the datafile), it would first be converted to hex, 1FBA00, then add two zeros at the beginning to bring the total number of characters to 8, then sent as '0' (30H), '0' (30H), '1' (31H), 'F' (46H), 'B' (42H), 'A' (41H), '0' (30H), '0' (30H), CR (0DH), LF (0AH). The letters in quotes are the characters you would send, the value in parentheses is the value of that character in hexadecimal notation.

Burn a program loaded in RAM into flash EEPROM

Why you may want to do it
For a program to run on power-up, it must be burned into the flash EEPROM of the TFX-11v2. Normally this is done from TFTools using the “Tattletale | Launch” menu selection and using the parallel port connection. If the parallel port is not available or you are not running TFTools.

How to do it

Host computer action	TFBASIC action
1) Send Ctrl-F	Echoes Ctrl-F
2) Send one's complement of Ctrl-F (F9 hex)	Burns program from RAM into flash EEPROM
3)	Returns '0' if successful Returns '1' if there was an error

Additional notes
The '#' prompt must be displayed by the TFX-11v2 before this protocol can begin.

Check if the datafile has been off-loaded using serial port commands

Why you may want to do it
This would be most useful to call before you erase the datafile. You would want to offer the option to off-load the datafile first.

How to do it

Host computer action	TFBASIC action
1) Send Ctrl-B	Returns '0' if datafile has been off-loaded Returns '1' if datafile has NOT been off-loaded Returns '2' if there was an error getting this information

Additional notes
The '#' prompt must be displayed by the TFX-11v2 before this protocol can begin.

Off-loading the datafile via the parallel port or the serial port will set the flag signaling that the datafile has been off-loaded. Writing to the datafile (via the STORE command) will clear the flag signaling that the datafile has NOT been off-loaded.

Read the version of TFBASIC using serial port commands

Why you may want to do it

How to do it

Host computer action	TFBASIC action
1) Send Ctrl-T	Sends MSB of Version (in binary)
2)	Sends LSB of Version (in binary)

Additional notes
The '#' prompt must be displayed by the TFX-11v2 before this protocol can begin.

The returned value is the same value that is returned by the TFBASIC read-only variable VERS. This is a binary value. For instance, if the TFBASIC version is 2.07, the value returned in 207. So, the most significant byte (MSB) returned would be 0 and the least significant byte (LSB) returned would be 207 (or CF hex).

Notice that only the major release and minor release are included in the returned value. For instance, for version 2.07, the major release is 2 and the minor release is 07. When TFBASIC first boots up, it may display a third number (the build number) if it is a beta release. That number will not be included in the version returned by this protocol.

Flush data to the EEPROM in a TFBASIC program

Why you may want to do it
When you STORE to the datafile, you are not writing directly to the EEPROM. Instead, the bytes are written to a RAM buffer. The TFBASIC buffer is 32 bytes wide. When this buffer fills, the 32 bytes are automatically written to the EEPROM. The buffer is also flushed when the STOP command is executed or when a Ctrl-C is detected at the serial port (and is not re-vectored with CBREAK) or if a “How?” error occurs (and is not re-vectored with ONERR). Other than this, the data remains in the buffer which can make some people nervous because if the TFX-11v2 loses power, the data in the RAM buffer is lost. You may want to flush the EEPROM buffer before doing a long SLEEP or HYB. You would want to flush the EEPROM buffer before reading data from the datafile. You would certainly want to flush the buffer before you execute a HALT command.

How to do it
There is a user-accessible assembly routine at hex address FD88 that flushes the buffer. The mnemonic that will be used in future versions of the manual will be FEESBF (flush EEPROM store buffer). You would use the CALL command like this:

```
call &hFD88,0
```

That’s all there is to it.

Additional notes
You should be able to call this function as often as you want but be aware of two things:

1) this function can use up to 15 milliAmps
2) this function can take up to 20 milliSeconds

User-accessible assembly routine used in the example

FEESBF Writes the contents of the 32-byte datafile buffer to flash EEPROM

Address: FD88

Change the value of DFPNT (datafile pointer) in a TFBASIC program

Why you may want to do it
Normally, the datafile pointer (DFPNT) is only changed when:

1) it is cleared to zero at program launch (via the parallel port)
2) it is cleared to zero if the TFX-11v2 loses power
3) it is incremented by the STORE command

DFPNT is a read-only variable in TFBASIC. If the programmer wants to save a section of EEPROM for later use or if the datafile is erased from within the program (see an example of this later in this document), you would want to change the value of DFPNT in the TFBASIC program.

How to do it
DFPNT, the address that will receive the next data from STORE, is located at addresses B0, B1, B2 and B3 hex except for TFBASIC version 2.00 through 2.07 where DFPNT was located at B2, B3, B4 and B5 hex. The Most Significant Byte is at the lowest address. Before changing the datafile address, though, you must flush the EEPROM buffer to the EEPROM (see the previous application note). Then the datafile pointer would be set by assigning the four-byte value to addresses B0 - B3 hex (or B2 - B5 hex for TFBASIC version 2.00 through 2.07). Finally, the new datafile pointer must be registered with the operating system by calling the user-accessible assembly function SBKADR (at address FD52 hex). This clears the EEPROM buffer and addresses that page on the EEPROM.

Now, the next STORE instruction will store data starting at the new datafile address. The value of DFPNT will be incremented automatically by STORE.

User-accessible assembly routine used in the example

FEESBF Writes the contents of the 32-byte datafile buffer to flash EEPROM

Address: FD88

SBKADR Sets up EEPROM store buffering system. Assumes the datafile storage pointer has been set to the correct value. Sets the address of the physical EEPROM and fills the EEPROM storage buffer with FF's.

Address: FD52

Notes: Because the FF's can safely be 'stored' over any pre-existing data in the datafile, this routine should work for any number in the datafile pointer. Power drain for the TFX-11v2 is higher when writing the EEPROM.

TFBASIC example for changing the datafile pointer

```
print dfpnt     // print the original datafile pointer
store 1,2,3,4   // store some data in the datafile
print dfpnt     // print the current datafile pointer
call &HFD88,0   // flush data in buffer to flash EEPROM (FEESBF)

input "Enter a new datafile pointer: "NewDFPointer

// check that the entered number is valid
if NewDFPointer < 0 | NewDFPointer > DFMAX
    print "Invalid datafile pointer"
    stop
endif

// location of datafile pointer was in a strange place for some versions
if VERS >= 200 & VERS <= 207
    pDFP = &HB5                    // DFPNT is at B2, B3, B4, B5 hex
else
    pDFP = &HB3                    // DFPNT is at B0, B1, B2, B3 hex
endif

// this section sets the new value in DFPNT
poke pDFP, NewDFPointer & &HFF      // set LSB of new DFPNT
pDFP = pDFP - 1                     // point to next byte of datafile pointer
NewDFPointer = NewDFPointer / 256
poke pDFP, NewDFPointer & &HFF      // set next byte of new DFPNT
pDFP = pDFP - 1                     // point to next byte of datafile pointer
NewDFPointer = NewDFPointer / 256
poke pDFP, NewDFPointer & &HFF      // set next byte of new DFPNT
pDFP = pDFP - 1                     // point to next byte of datafile pointer
NewDFPointer = NewDFPointer / 256
poke pDFP, NewDFPointer & &HFF      // set MSB of new DFPNT

call &HFD52,0                       // set up buffer and address for future STORE (SBKADR)

print dfpnt                         // print the updated datafile pointer
stop
```

Right-click paperclip icon and Open or Save example program

Erase datafile pages from within a TFBASIC program

Why you may want to do it
The two main reasons for erasing pages of the datafile from within a TFBASIC program are:

1) You want to have a wrapping datafile. That is, when you reach the end of the datafile, you want to start storing at the beginning of the datafile again. In this way, you can create an "endless" datafile where you only store the latest information and erase data that is too old.

2) You want to off-load your data using your own method and then erase the entire datafile so you can start storing data again. This would be a useful method to never have to stop your program and wait for input from a desk-top computer.

How to do it
The main function is ERASEP. This does the actual erasing but you first need to flush the EEPROM buffer to the flash device by executing FEESBF. Then call ERASEP to erase each EEPROM page. After the last call to ERASEP, you may want to change the value of the datafile store pointer (DFPNT). If you have erased the entire datafile, you will want to reinitialize the datafile

Detailed notes
Function ERASEP requires a page number argument. A "page" is the smallest number of bytes that can be erased on a flash memory part. Currently, the flash on the TFX-11v2 has a page size of 512 bytes but this could change in the future. To get the page number of the beginning of the datafile, read the two bytes at absolute addresses AE - AF hex (the byte at AE hex is the most significant byte). NEVER pass a page value less than this to the ERASEP function or you will erase a page of the TFBASIC operating system! Another piece of information that is important is the page number of the end of the datafile. This is found at absolute addresses 125 - 126 hex (the byte at 125 hex is the most significant byte). This is the highest page number that you can erase. Passing a number higher than this value is undefined.

This operation will not change the datafile pointer used for storing. The datafile pointer may need to be updated. See the section titled "Change the value of DFPNT in a TFBASIC program". Also, this new value will need to be registered and the RAM buffer prepared by calling the function SBKADR (address FD52H) explained later.

An important note about the consequences of erasing the datafile yourself:
There are two flags involved with the datafile that you need to consider. One is the DFERASED read-only variable and the other is an internal "Datafile Off-loaded" flag. You don't have direct control over either of these flags. They should both be cleared after you off-load the datafile and then erase the entire datafile. The only ways to set these flags correctly is by either doing the off-load and erase functions via the TFTools program or, if you do the off-load and erase yourself, by calling the RSETDF Assembly Language Subroutine documented below. This subroutine is *only* available in TFBASIC version 2.08 and above. Otherwise, you have to live with the consequences of erasing the datafile but not setting the DFERASED flag nor clearing the internal "Datafile Off-loaded" flag.

User-accessible assembly routines used in the example

FEESBF Writes the contents of the 32-byte datafile buffer to flash EEPROM

Address: FD88

ERASEP Erase a flash EEPROM page. Page number number to erase in A/B registers With the Most Significant Byte in the A register.

Address: FD85

Notes: Beware that ANY page of the flash EEPROM can be erased with this routine. Even the Header Page or a page of TFBASIC or the user's program can be erased.

Power drain for the TFX-11v2 is higher when erasing the EEPROM.

SBKADR Sets up EEPROM store buffering system. Assumes the datafile storage pointer has been set to the correct value. Sets the address of the physical EEPROM and fills the EEPROM storage buffer with FF's.

Address: FD52

Notes: Because the FF's can safely be 'stored' over any pre-existing data in the datafile, this routine should work for any number in the datafile pointer. Power drain for the TFX-11v2 is higher when writing to the flash EEPROM.

RSETDF Resets the datafile pointer (DFPNT) to zero, initializes the datafile buffer and sets both the DFERASED flag (signaling that the datafile has been erased) and the internal OFF-LOADED flag (signaling that the datafile has been off-loaded, which really means there is no data to be off-loaded yet).

Address: FD4F

Notes: This function should ONLY be called when you have erased the entire datafile using the methods in the "Erase the datafile from within a TFBASIC program" section of this document.

TFBASIC example for erasing the entire datafile

The following example erases the entire datafile. Because of this, the example also calls RSETDF to reset the datafile handling code. If you are only deleting part of the datafile (as you would if you just want to overwrite old data), *do not* call RSETDF.

```
print "Erasing datafile (takes about 25 seconds)"
asm $
FEESBF  equ H'FD88  ; flush data in buffer to flash EEPROM
ERASEP  equ H'FD85  ; function to erase a page of flash EEPROM
SBKADR  equ H'FD52  ; function to set up buffer and its address
RSETDF  equ H'FD4F  ; function to reset datafile pointer, buffer and flags
DFFirst equ H'AE    ; two bytes contain first datafile page number
PgCount equ H'125   ; two bytes contain number of pages

        jsr FEESBF  ; first, flush any data in buffer to flash EEPROM
        ldx DFFirst ; X register holds datafile page number
_EEClr  cpx PgCount ; check if this is end of datafile
        bcc _xeec   ; branch to _xeec if end of datafile, ie >= PgCount
        pshx        ; save page number
        xgdx        ; ERASEP expects page number in A/B registers
        jsr ERASEP  ; erase the flash page
        pulx        ; retrieve page number
        inx         ; point to next datafile page
        bra _EEClr  ; loop back to do this page
_xeec   rts
        end
print "Datafile erased"

if VERS > 207       // a new user assembly routine was added with version 2.08
    call RSETDF,0   // this does everything in the following ELSE statement and more
else    // otherwise, older versions must do all this stuff
    // clear datafile pointer to zero
    if VERS >= 200 & VERS <= 207 // location of df pointer different in some versions
        pDFP = &HB2
    else
        pDFP = &HB0
    endif

    poke pDFP, 0        // set MSB of new DFPNT
    pDFP = pDFP + 1     // point to next byte of datafile pointer
    poke pDFP, 0        // set next byte of new DFPNT
    pDFP = pDFP + 1     // point to next byte of datafile pointer
    poke pDFP, 0        // set next byte of new DFPNT
    pDFP = pDFP + 1     // point to next byte of datafile pointer
    poke pDFP, 0        // set LSB of new DFPNT
    call SBKADR,0       // set up buffer for further STOREs
endif

stop
```

Right-click paperclip icon and Open or Save example program

Prevent Corrupt Datafiles after a Power Failure

It is possible to see corrupted data in the datafile of a TFX-11v2 after a power failure or after resetting the TFX-11v2 by using the RESET line (pin J3-12).

The datafile pointer is volatile
This all comes from the fact that the datafile pointer in TFBASIC is a volatile value - whenever power is lost, the value is lost and on next start-up will be cleared to zero. This seems strange because the datafile itself is non-volatile. The reason is that the flash EEPROM, used to store the datafile and make it non-volatile, has a limited write capacity. After 100,000 or more erase-write cycles, the flash EEPROM cells begin to fail. This is fine for data because you will probably never write to any one location more than that over the lifetime of the TFX-11v2. But, if we wrote the current datafile pointer to flash EEPROM every time it was changed, you would ruin those few flash cells by the time you had filled the datafile once.

So, the datafile pointer must be stored in RAM and when power is lost or if the TFX-11v2 is reset, the datafile pointer is cleared to zero. If you've previously stored data in the datafile, your next write to the datafile will be writing to memory that already has data stored in it. You might think that you will just overwrite the old data with new data but that is incorrect.

Overwriting data corrupts the data
Flash EEPROM is not like normal memory. It must be erased before you store data. The reason for this is that flash EEPROM memory acts like a series of fuses. Erasing the memory resets (or closes) all the fuses and when you read them, they read as a 1. When you store data to a flash EEPROM, any bits that are 1 just leave the fuse in its closed state. Only bits that are 0 really change anything - a bit that is 0 opens (or burns) the fuse. Then when you read data back, fuses that are closed read as 1 and fuses that are burned (or open) read as 0. One of the limitations of flash EEPROM is that you cannot write a 1 into a cell that already has been written to 0 (open or burned). The ONLY way to close the fuse again is to erase the flash and this must be done in pages (or groups) of bytes. In the case of the TFX-11v2, this page size is 512 bytes.

An example
If you write the byte value 72 hex to a flash memory, the eight cells that comprise this byte look like:

0 1 1 1 0 0 1 0

where each 0 is an open (or burned) memory cell and each 1 is a closed (thus, unchanged) memory cell. If we now try to write the byte value 94 hex to this same flash memory byte, you want to write:

1 0 0 1 0 **1** 0 0

but you can't change the left-most bit (in bold face) back to a 1 - it's a burned fuse. Likewise, you can't change the third bit from the right (in bold face) back to a 1. The only way to change a 0 bit to a 1 bit is to erase the entire 512 byte page. If we do write 94 hex to this byte without erasing it first, only bits that are being changed to 0 will be written. So, we'll end up with:

0 0 0 1 0 **0** 0 0

Notice again the bits in bold face. If we read this data, it reads as 10 hex. It looks like garbage because it doesn't look like either of the bytes we wrote to that location. When the datafile pointer gets reset to 0, and you write data starting again at 0, you are only writing new 0 bits to your data and making it look like garbage.

Protect your data
We've included a way to keep this from happening. There are two read-only values available to your TFBASIC program that you need to check when your program starts and before you try to store data to the datafile. Read DFPNT and DFERASED. If DFERASED is true (or non-zero), it means the datafile has

been erased and it is safe to write to the datafile. If DFERASED is false (or zero) and DFPNT is 0, it means you are going to be storing data over previously stored data. You have two options at this point:

1) Stop your program, off-load the data and erase memory OR

2) Find where your previously stored data ends and set the datafile pointer to one byte beyond that and continue with your program

Option 1 is simple and requires no more explanation. Option 2 is more complex. If you find that DFPNT and DFERASED are both 0 (meaning you will corrupt data if your use the STORE command), read the datafile bytes (starting at 0) until you find a block of bytes (a record) that read as FF hex - meaning they are erased. Then you have to set the DFPNT value to the beginning of the erased area. We have included a program that does this for you. The tricks to using this are that you must supply a record length to the program (that is, how many bytes are stored each time you call the STORE command) and there must be at least one non-FF hex byte in every record you write. Try running the example program. It works best if you Launch the program (select the Launch item under the Tattletale menu in TFTools) so that the program runs automatically on power-up. After the program ends, cycle the power so the program starts again. Do this a few times so you see how the program works. Notice that this program has a record length of 132 bytes. Your program will probably be different.

Real data, not garbage

Making sure you are not overwriting flash EEPROM bytes that have already been used is critical to your data collection efforts. The steps outlined above should help you ensure that your data is valid.

TFBASIC example for finding erased portion of datafile

```
print "Program to find where erased datafile starts"
RecordLength = 132                  // store 4 bytes 33 times
print "DFPNT = ",DFPNT
print "DFERASED = ",DFERASED
print "RecordLength= ",RecordLength

if (DFPNT = 0) & (DFERASED = 0)     // check for trouble
    print "Searching for erased area of datafile"
    gosub FindErasedDF
    NewDFPointer = iErasedArea
    gosub SetDFPointer
    print "Updated DFPNT = ",DFPNT
endif
if (DFPNT >= 0) & (DFPNT < DFMAX+1) // if datafile pointer is valid
    print "Storing data"
    for i = 1 to 33
        store #4,i
    next i
else
    print "Unable to find an erased area"
endif
stop

//*********************************************************************
// FindErasedDF
//
// Returns with iErasedArea set to start of erased area in datafile or
// equal to DFMAX + 1 (impossible datafile address) if datafile is
// entirely filled
//
// Before calling this function, RecordLength MUST be set to the number
// of bytes you store to the datafile in each record.
//*********************************************************************
FindErasedDF:
    iErasedArea = DFMAX+1   // start by assuming we can't find an erased area
    iGetPointer = 0         // start looking at beginning of datafile area
    iCount = RecordLength   // the length of data we store, in bytes

    while iGetPointer <= DFMAX
        if get(iGetPointer) = &HFF          // in an erased area
            if iErasedArea = (DFMAX + 1)    // if first byte of erased area...
                iErasedArea = iGetPointer - 1
            endif
            iCount = iCount - 1
        else                                // NOT in an erased area
            iErasedArea = DFMAX + 1         // mark that we haven't found it
            iCount = RecordLength           // reset erased byte count
        endif
        if iCount = 0
            return
        endif
        if (iGetPointer % 1024) = 0 print iGetPointer,\13;
    wend
    return
```

```
//*********************************************************************
// SetDFPointer
//
// This subroutine will set the datafile pointer (DFPNT) to the value
// in NewDFPointer. The subroutine checks that NewDFPointer is a valid
// number from 0 to DFMAX. If the value is not valid, this subroutine
// returns without doing anything (except printing an error message).
//*********************************************************************
SetDFPointer:
    if NewDFPointer < 0 | NewDFPointer > DFMAX
        print "Invalid datafile pointer"
        return
    endif

    call &HFD88,0                  // flush data in buffer to EEPROM (FEESBF)

    // location of datafile pointer was in a strange place for some versions
    if VERS >= 200 & VERS <= 207
        pDFP = &HB5
    else
        pDFP = &HB3
    endif

    poke pDFP, NewDFPointer & &HFF  // set LSB of new DFPNT
    pDFP = pDFP - 1                 // point to next byte of datafile pointer
    NewDFPointer = NewDFPointer / 256
    poke pDFP, NewDFPointer & &HFF  // set next byte of new DFPNT
    pDFP = pDFP - 1                 // point to next byte of datafile pointer
    NewDFPointer = NewDFPointer / 256
    poke pDFP, NewDFPointer & &HFF  // set next byte of new DFPNT
    pDFP = pDFP - 1                 // point to next byte of datafile pointer
    NewDFPointer = NewDFPointer / 256
    poke pDFP, NewDFPointer & &HFF  // set MSB of new DFPNT

    call &HFD52,0                  // set up buffer and address for STORE
    return
```

Right-click paperclip icon and Open or Save example program

List of additional Assembly Language Subroutines

Since publication of the TFX-11v2 manual, some more Assembly Language Subroutines have been made available for use in your own assembly language routines or by using the TFBASIC CALL command. See page 167 for information on Assembly Language Subroutines and page 61 for information on the CALL command. The following locations are the entry points into fixed assembly language routines. The value at the beginning of each function's description is the address of that function; this address is the argument to the CALL command.

RSETDF **FD4F**. This function resets the datafile pointer (DFPNT) to zero, initializes the datafile buffer and sets both the DFERASED flag (signaling that the datafile has been erased) and the internal OFF-LOADED flag (signaling that the datafile has been off-loaded, which really means there is no data to be off-loaded yet). This function should ONLY be called when you have erased the entire datafile using the methods in the "Erase the datafile from within a TFBASIC program" section of this document.

SBKADR **FD52**. Sets up the flash EEPROM STORE buffering system. Assumes the datafile storage pointer has been set to the correct value as outlined in the "Change the value of DFPNT in a TFBASIC program" section of this document. This function sets the address of the physical flash EEPROM and fills the EEPROM storage buffer with FF's. This function should only be called when the datafile STORE pointer (DFPNT) has been specifically changed by you.

SDIF **FD70**. Read up to 32 bits as clocked serial data using the same pins as the TFBASIC SDI command. Before calling this function:

X register Must point to a 4-byte variable that will receive the data.
This variable's initial value must be zero. It will contain the input data when the function ends.

B register Must contain the number of bits to read.
This value must be >= 1 and <= 32.

A register Must contain the mode of the transfer
The A register mode value must be 0, 1 or 2:

if A = 0, the transfer is like the TFBASIC SDI command, there is one less clock pulse than the number of bits transferred.

if A = 1, there will be an extra clock pulse before reading the first bit followed by the regular clock pulses before each subsequent bit that is transferred.

if A = 2, there will NOT be a clock pulse before the first bit but there will be an additional clock pulse after reading the last bit.

SDOF **FD73**. Write up to 32 bits as clocked serial data using the same pins as the TFBASIC SDO command. Before calling this function:

X register Must point to a 4-byte variable holding the data to send.

B register Must contain the number of bits to write.
This value must be >= 1 and <= 32.

SDOCH **FD76**. Write the 8 bits in the A register as clocked serial data using the same pins as the TFBASIC SDO command.

ERASEP **FD85**. Erase a flash EEPROM page. The page number to erase must be in the A and B registers (Most Significant Byte in the A register). Beware that ANY page of the flash EEPROM can be erased with this routine. Even the Header Page or a page of TFBASIC or the user's program can be erased. Read the "Erase datafile pages from within a TFBASIC program" section of this document for more information.

GETMEM **FD91**. Read one byte of data from the datafile at address held by the Internal Datafile Read Pointer and return the value in the B register.

The value of the Internal Datafile Read Pointer can either be explicitly set or the current value can be used. On power-up, or when a program starts, the Internal Datafile Read Pointer is cleared to zero. After using the TFBASIC GET or GETS commands, the Internal Datafile Read Pointer will point to the byte after the data that was read by GET or GETS command. The internal datafile read pointer is located at addresses B4, B5, B6 and B7 hex except for TFBASIC version 2.00 through 2.07 where the Internal Datafile Read Pointer was located at B6, B7, B8 and B9 hex. The Most Significant Byte is at the lowest address. To explicitly set the internal datafile read pointer, simply write the value to the four bytes listed here. Then call GETMEM. The value of the Internal Datafile Read Pointer will automatically increment by one after GETMEM returns.