CHAPTER 18. DISK FILE UTILITY

The source code in this chapter is in UTILITY.BLK, screens 4, 7 and 8.

18.1. DISPLAYING SCREENS IN A FILE

The editor allows us to examine and modify screens, one at a time. For large application which requires many screens, it is necessary to have some commands which allows us to scan the contents of a file so that we will know which screen to examine in detail. These commands are also useful in generating hardcopy of source code on a printer.

```
: .SCR              ( --- )       Print the current screen number and the file name.
        ." Scr # " SCR ?          Print the screen number,
        8 SPACES FILE?            and the file name.
        ;

: LIST              ( n --- )List the specified screen in the 16 by 64 character format.
                    Pressing a key stops the printing.  LIST also makes n the
            current screen.
        1 ?ENOUGH                 Make sure n is on the stack.
        CR DUP SCR !              Make n the current screen.
        .SCR                      Print screen number and file name.
        L/SCR 0 DO                Scan 16 lines.
                CR I 3 .R SPACE   Print the line number first.
                DUP BLOCK         Get the screen from the file and return the buffer address.
        I C/L * +           Line address in the buffer.
                -TRAILING >TYPE   Print one line of text.
                KEY? ?LEAVE       Quit if a key is pressed.
        LOOP  DROP CR             House keeping.
        ;
```

To print multiple screens on paper, it is nice to arrange three screens to a page. By convention, the first screen number is a multiple of three so that a group of three screens forms a unit in arranging your source code. TRIAD is the command to print screens in this style.

```
: TRIAD             ( n --- )Print three screens on a page. The nth screen must be
                    printed. The top screen has a screen number modulo 3.       12 EMIT
            Form feed.
        3 / 3 *                   Modulo 3 boundary.
        3 BOUNDS DO               Print only 3 screens.
                I LIST          One screen at a time.
        LOOP ;
```

The top line in a screen is usually a comment line. Besides the documenting function, it also allows you to scan a range of screens and identify the contents of screens easily. F83 also puts an ID stamp at the

end of the top line. The command INDEX prints the top lines of a range of screens. It is very handy as a substitute of a screen directory.

```
: .LINE0            ( n --- ) Print line 0 of nth screen.
        DUP 3 MOD               If n is evenly divisible by 3,
        0= IF CR THEN           send out an extra line feed.
        CR DUP 3 .R SPACE Print the screen number.
        BLOCK                   Get the screen from file.
        C/L -TRAILING >TYPE     Print the top line.
        ;

: INDEX     ( start end --- )       List the top lines in a range of screens.
        2 ?ENOUGH                   It needs two parameters.
        1+ SWAP DO          Scan the range of screens.
            I .LINE0                Print top line only.
        LOOP  CR    ;

: IND               ( n --- ) Start printing index lines from screen n.  Continue until a
                    key is pressed.
        BEGIN                   Start with screen n.
            DUP .LINE0          Print one top line.
            1+                  Increment n .
        KEY? UNTIL          Stop when a key is pressed.
        DROP ;
```

18.2. DISK BUFFERS

Screens or blocks of 1024 bytes are the basic units for FORTH program source code. The size of block is convenient to develop modularized programs because the block of text fits comfortably inside a standard CRT screen while allow enough room to do loading and testing of the source code. Since the screens can be compiled independently, it is the common practice to use a load screen which loads other screens in the order that is required by the application. Therefore, screens do not have to be arranged in specific order. However, there are times that we would like to move texts screens around and physically arrange them in some order, especially if the texts are to be printed and communicated to other people. Disk copying utility in F83 enables the user to copy single or multiple screens within a file and also from one file to another.

To fully appreciate the disk copying utility in F83, it is necessary to understand fully the disk buffer structure used in the F83 system to handle the traffic from and to the disk files. Let us briefly review the F83 disk buffers.

In the high RAM memory, a number of disk buffers are assigned by the system at boot up time. Each buffer is 1024 bytes in size to hold a block of text or data, corresponding to the data stored in a corresponding block in a disk file. The number of disk buffers is specified in a constant #BUFFERS. Immediately before the first disk buffer, there is an array storing the essential information about the disk buffers. Each disk buffers has a corresponding entry in this array. An entry consists of four cells for the block number, a pointer to the parent file, the address of the disk buffer, and an update flag. Whenever a block is accessed, its array entry is moved to the beginning of the array, indicating it is the most recently accessed buffer. The buffer with an array entry at the end of the array is the least accessed buffer and will be re-assigned to receive other disk block when necessary.

In doing multiple block copying, it is generally desirable to read/write all the disk buffers together and in sequence to minimize the disk head movements and the start/stop of the disk drive. F83 disk copy utility tries to optimize the disk accessing, as it normally uses four disk buffers.

More detailed information on the disk I/O is covered in the chapters dealing with the nucleus and DOS of the F83 system.

18.3. SINGLE BLOCK COPYING

To copy one block of text or data to another block in the same disk file, the most efficient way is not physically copying the block, but to bring the source block into one disk buffer and reassign that buffer to the destination block with the update flag set. The data in the buffer will be flushed into the destination block when the buffer is needed for other disk I/O transaction.

```
: ESTABLISH ( n --- )       Set the block number of the most recently referenced block
                            to n, thus assign the buffer to the nth block.
        FILE @ SWAP             Get the current file fcb.
        1 BUFFER#               Get the address of the first entry in the buffer pointer array.
        2!                      Store n into the first cell in that entry, forcing the most
                                recently referenced block to become the nth block. The
                                current file number is store into the second cell.
        ;

: (COPY)        ( from to --- ) The primitive word to copy one block to another.  OFFSET  @
                Get the block offset number of the current file.
        +                       The actual block number of destination.
        SWAP                    Get the source block number to top of stack.
        IN-BLOCK                Get the source block into the most recently referenced
                                buffer.
        DROP                    The buffer address is not needed.
        ESTABLISH               Claim this buffer for the destination block.
        UPDATE                  Update the buffer so that it will be written back to the
                                destination block.
        ;

: COPY          ( from to --- ) To be careful, copy a block and explicitly flush it to disk.
        FLUSH                   Empty the disk buffers to disk files.  Important when
                                accessing multiple files.
        (COPY)                  Do the copying.
        FLUSH                   Flush the destination block.
        ;
```

18.4. MULTIPLE BLOCK COPYING

When we copy a range of consecutive blocks from one place to another, complication arises if the destination range overlaps with the source range. To avoid conflicts in the overlapped copying, the direction of copying must be carefully selected.

```
VARIABLE HOPPED         The number of screens to skip when copying.
VARIABLE U/D            The direction of copying to avoid overlap.
```

DEFER CONVEY-COPY	A deferred word. It will be used to copy within one file and copy between files.
' (COPY) IS CONVEY-COPY	For the moment, define it to copy blocks in the same file.

```
: HOP           ( n --- )       Specify the number of screens to skip in copying.  HOPPED  !
                Store n in HOPPED.
        ;

: .TO           ( n1 n2 --- n1 n2 )     Print a message while copying.
        CR      OVER .                  Print n1.
        ." to "  DUP .                  Print n2.
        ;

: (CONVEY)  ( blk n --- blk+-n )    Move a range of screens in the direction specified by U/D.  0  ?DO
                        Copy n blocks.  If n=0, exit immediately.
                KEY? ?LEAVE         Stop if user hit a key.
                DUP DUP             Source block number.
                HOPPED @ +  Destination block number.
                .TO                 Print something to indicate that the computer is working
                        hard.
                CONVEY-COPY         Copy one block a time.
                U/D @ +             The next source block.
        LOOP    FLUSH               Flush the blocks still in the disk buffers.
        ;

: CONVEY    ( first last --- ) Move a set of screens.  First determine the direction of
                copying to prevent overlap.  Move the blocks as a set whose
                size is determined by the number of available disk buffers.  FLUSH
                Clear the buffers.
        HOPPED @                    Get the screen number to skip.
        0< IF                       If copying from high block to low block,
                1+ OVER -           Number of blocks to copy.
                1                   Copy in the forward direction.
        ELSE                        Copying from low block to high block.
                DUP 1+ ROT -        Number of blocks to copy.
                -1                  Copy in the backward direction.
        THEN    U/D !       Store the direction code into U/D.
        #BUFFERS /MOD               Groups of blocks to be copied together.
        >R (CONVEY) R>              Copy the remainder blocks which do not fill the pipeline.
        0 ?DO                       Pipe the rest of block through all the buffers.         #BUFFERS
(CONVEY)    Copy #BUFFERS blocks all at once.
        LOOP    DROP                Leftover block number.
        ;
```

If you know the destination block number and do not want to use HOP to specify the number of blocks to be skipped, you can use the following commands to do the copying:

```
        <first>  <last>  TO  <destination>  CONVEY
```

where 'first', 'last', and 'destination' are block numbers indicating the range of source blocks and the first destination block number.

```
: TO        ( first last --- first last )   Calculate the blocks to be hopped and store the
                                            number in HOPPED.
    SWAP                Get 'first' to top of stack.
    BL WORD                 Read the next number.
    NUMBER DROP             Convert the number to an integer.
    OVER -                  The hopping distance.
    HOP                     Store it in HOPPED.
    SWAP                Restore the 'first last' order on data stack.
    ;
```

18.5. MULTIPLE FILE BLOCK COPYING

F83 Version 2.0 and above was modified so that when screens are copied, it is always read from the in-file and written to the current file. Therefore, it is not necessary to define special commands to copy screens from one file to another. You must remember that when you OPEN a file, you assign the file to both the current file and the in-file. If you open another file using the FROM command, this file becomes the in-file which is always to be read. In the early versions of F83 in which the role of in-file was not clearly defined, many block copying commands had to be redefined in the FILES vocabulary. The following discussion applies only to the F83 Versions 1.x.

One of the vantages in using the DOS operating system to host the FORTH system is that we can organize the disk storage into named files which help us to using the disk storage more conveniently by grouping related screens into separated files. However, it is often necessary to transfer common utility from one file to another. The multiple file screen copying utility in F83 system makes it easy to copy either single screen or a range of screens from one file to another by new versions of COPY and CONVEY. These commands are redefined in the FILES vocabulary so that they can be accessed independent of the existing COPY and CONVEY commands in the FORTH vocabulary.

```
ONLY FORTH ALSO FILES DEFINITIONS       Make FORTH and ONLY the resident
                                        vocabularies and FILES the context and current vocabulary.

: COPY      ( from to --- )         Copy a screen from the FROM file to the current file. The
                                    FROM file must be declared by the FROM xxx commands,
                                    where xxx is the FROM file name.
    SWAP            Get the source block number to top of stack.
    EXCHANGE            Exchange the FROM file with the current file.
    BLOCK               Read the source block from the FROM file.
    SWAP            Get the destination block number to top of stack.   EXCHANGE
        Restore the current file.
    BLOCK               Get the destination block from the current file.
    B/BUF CMOVE         Copy the contents of the source screen into destination
                        screen buffer.
    UPDATE              Update the destination block so it will be copied back to the
                        current file.
    ;
```

```
: CONVEY            ( first last --- ) Copy a range of screens from the FROM file to the current
                                       file. The screen range is the current file is offset from that in
                                       the FROM file by the number in HOPPED.
        ['] CONVEY-COPY >BODY Get the parameter field of the word CONVEY-COPY.
        @                             The execution address of (COPY).
        >R                            Save it on the return stack.
        ['] COPY                      Execution address of the COPY in the FILES vocabulary,
                                       which does copy between files.
        IS CONVEY-COPY                Vector CONVEY-COPY to the COPY we just defined
                                       above.
        CONVEY                        Now, CONVEY will copy a range of screens from FROM
                                       file into the current file because CONVEY-COPY is vectored
                                       to the new COPY command.
        R> ['] CONVEY-COPY
        >BODY !                       Restore CONVEY-COPY to the old single file COPY.      ;
```

This is an example in using a deferred word to do different thing by vectoring it to different commands. The definitions using the deferred word do not have to be change at all.