Trident disk for the Alto

by R. D. Bates

June 15, 1979 (revised November 24, 1979)

This document describes the Trident disk controller and microcode interface which is available for placing high performance disk capabilites on Alto computers.

Key words and phrases: Trident, Alto disk, TFS.

XEROX
PALO ALTO RESEARCH CENTER
3333 Coyote Hill Road / Palo Alto / California 94304

This memo describes the facility for an 80 megabyte capacity, 9.7 megabit transfer-rate disk for use on Alto computers. The new disk is implemented *in addition* to the standard Alto disk. This simply means that new microcode tasks (one for output and one for input) are used, leaving the standard disk microcode intact. The requirements for implementing this facility are a Trident T-80 disk ($5500), a disk control board ($800), and an Alto II. Older style Altos can have Trident disks but there controller is not readily avalable.

1. The basic disk system

The disk controller is designed to drive any member of the **Trident** family of disk drives manufactured by Century (now part of Xerox). Currently the T-80 and the T-300 drives have been connected. A summary of the specifications for the **T-80** drive is found in Fig. 1.

The disk system is accessed through a many level addressing scheme. First a particular disk drive is accessed (there may be from 1 to 16 drives attached to a single disk controller). The surface of a particular disk pack is selected by specifying 1 of 5 *heads*, and 1 of up to 815 tracks. Each track is further broken down into sectors. The number of sectors is determined by jumpers within the disk drive which determine the number of words reserved for each sector (3070 words max.). Each sector can be further broken down into blocks, each of which can be either read, written, or checked. Reading or writing within a sector must start with the first block and continue, but one does *not* have to read or write to the end of a sector. A block may not be read after writing any block within that sector. The checking scheme is intended for checking the header and label, and will inhibit writing later blocks within the sector should a compare error occur.

The size and number of blocks within any sector is determined by parameters from the Alto program which are given to the microcode for each sector. Thus the sector formatting does not have to be the same for the whole disk. All the necessary delays for turning on read or write circuitry are created within the disk controller with a single PROM which is programmed with the appropriate delays. The choice of the number of blocks, and their size, should be made (unless you wish to defeat the "sector overflow" check) such that all blocks are within a given sector area on the disk.

Due to the high data density used in this system, a disk pack certified for use by these disk drives does not have to be completely perfect. A disk pack is suitable if it has no more than three bad areas on any of the five surfaces; where a bad area is defined as one which could potentially cause read errors of no more than 11 data bits in length. In order to correct such errors, as well as other very occasional read errors, a scheme of error detection and correction has been implemented in the disk controller which will detect (with very high probability) errors of any length, and correct any burst error of 11 bits or less. Warning! If a burst error of more than 11 bits occur there is a significant possibility that the error correction algorithm will *incorrectly* correct the error and thus double the number of bad bits.

T-80 SPECIFICATIONS AND CHARACTERISTICS

CAPACITY

82.1 million 8-bit bytes - unformatted (312 for T-300)

TRANSFER RATE

9.67 megabits per second
16 bit word every 1.65 μs.

ACCESS TIME

Track to Track Positioning - 6 milliseconds maximum
Average Positioning - 30 milliseconds
Full Stroke Positioning - 55 milliseconds maximum
Average Latency - 8.3 milliseconds

ROTATIONAL SPEED

3600 revolutions per minute (16.66 milliseconds per revolution)

PACK START/STOP TIME

Start Time - 20 seconds
Stop Time (with dynamic braking) - 20 seconds

SECTOR LENGTH SELECTION

12 bit increments through jumpers on sector board

DENSITIES

Track Density - 370 tracks per inch
Recording Density - 6060 bits per inch maximum

DISK PACK CHARACTERISTICS

Disk Pack - IBM 3336-type components
Recording Surfaces - 5 plus 1 servo surface
Tracks per surface - 815

OPERATING METHODS

Recording Method - Modified Frequency Modulation
Positioning Method - Linear Motor; Track Following Servo

MECHANICAL SPECIFICATIONS

Size - 17.8" wide x 10.5" high x 32" deep
Weight - 230 pounds

ERROR RATE

Recoverable: 1 error in 10^{10} bits
Non-recoverable: 1 error in 10^{13} bits
Positioning: 1 error in 10^{6} seeks

CONTROLS AND INDICATORS

Ready Indicator
Fault Indicator
Start/Stop Switch
Read Only Switch
Degate Switch (takes disk off-line for testing)

FIGURE 1

2. Available software

A file system equivalent to that used on the standard Diablo 31 disk drives is available. This system is described in the Alto Operating System Reference Manual. In addition, other programs associated with or for use on the Trident disk system are as follows:

[MAXC]<ALTO>TFS.dm
: This file package is equivalent to the "BFS" file system in use on the Alto which are used for implementing file oriented access the disk. This also contains the microcode necessary for driving the disk controller, including a small routine called during error recovery. Documentation for the TFS package (and the TFU program, mentioned below) is contained in the Alto Subsystems manual and in <AltoDocs>TFS.tty.

[MAXC]<ALTO>TRIEX.run (also available as a network boot file)
: An exerciser program used for basic debugging and to check general disk reliability.

[MAXC]<ALTO>TFU.run
: A program for creating and manipulating a file system on a Trident disk, for certifying new disk packs for operation, and for exercising the file system at a high level.

It should be mentioned that writing software to drive the Trident controller directly has proven to be quite difficult, especially in the areas of initialization error recovery. If you choose to run the Trident controller directly (through the interface described in section 4 of this document), you should first examine carefully the code in the TFS package.

3. A "standard" configuration

The following is a description of the sector formatting which is most often used for general file storage on this disk system. If you wish to look at other configurations, you can go to the section on Format Specifications for full details. JUMPERS ON TRIDENT CARD!!

Each track (10,080 words) is divided into 9 sectors of 1120 words each. Each sector is then divided into 3 blocks in much the same way as the standard Alto disk. The first block is a Header, and contains 2 words for identifying the disk address of that particular sector. The Header is normally read and checked by the controller in order to verify that the disk drive is actually where the controller thinks it is. The second block is the Label, and contains 10 words for storing useful file dependent information, such as name, type, address of next sector, etc.. The last block is 1024 words of actual file data.

Using the above format on a T-80 drive yields a total formatted capacity of 37,552,200 words (1024 words x 9 sectors x 815 cylinders X x surfaces). Using this format, the average transfer rate per disk revolution is 8.85 megabits per second.

4. Command blocks

The disk controller contains a "Run-Enable" flip-flop (initially turned off) which can be controlled by the emulator task through the execution of the SIO (StartIO) instruction. An SIO with bit 10

set will cause Run-Enable to be set. An SIO with bit 11 set to one will cause Run-Enable to be reset. Note that issuing an SIO with bit 10 set will wake up the microcode once and thus may report status in the absence of sector pulses from a disk. This facility is important since sector pulses are present only when a drive is running with the heads loaded.

The Alto program communicates with the disk controller via a four-word block of main memory at location KBLK (currently 640 octal), specified as follows:

KBLK	Pointer to first disk command block
KBLK+1	number of currently selected drive
KBLK+2	cylinder position used for the last command
KBLK+3	Status of current drive at last sector pulse
KBLK+4	Error which caused last transfer to be aborted

The microcode program is woken up at every sector pulse by the hardware. The program then updates the status entry, and checks to see if the command pointer is non-zero. If a command is present then processing begins, otherwise the microcode blocks until the next sector pulse. The command table pointed to by KBLK is a variable length block which completely specifies a transfer for a given sector. The block has the following format:

DCB	Cylinder select
DCB+1	left byte - Head select (values 0 through 4)
DCB+1	right byte - Sector count (values 0 through 15)
DCB+2	Drive select
DCB+3	Pointer to next command block
DCB+4	Disk Command Seal (octal 122645)
DCB+5	R/W command
DCB+6	Word count
DCB+7	Memory block pointer
DCB+8	Error Correcting Code 0
DCB+9	Error Correcting Code 1
DCB+10	Status
DCB+11-	
DCB+16	Possible repeats of DCB+5 through DCB+10
-	-
-	-
DCB+?	All zero's R/W command word to terminate
DCB+?+1	Interrupt word

The disk command block is made up of a five word introduction followed by a variable number (0 to n) of six word block descriptors. The introduction contains a description of the sector location, and the block descriptors describe the transfer required for each block within the sector. Processing terminates when a R/W command within a block descriptor is found to be zero: at this time, the following word is ORed into NWW to generate an interrupt on the channels corresponding to one bits in that word.

A detailed description of the entries in these two tables follows.

KBLK -Command Block Address-

This word is tested by the microcode every sector time or upon the completion of a disk command block. If the value is zero, then the current disk status will be placed in KBLK+3 and the microcode will block until the next sector pulse. If the value is non-zero, then command block proccessing will start at the address pointed to by KBLK. The microcode will update KBLK with the address of the command block it is currently working on. However, the microcode may advance to the next command block before all data transfer activity for the previous one has completed: consequently, checking for KBLK to be zero is *not* a safe way to determine whether the disk is idle.

KBLK+1 -Drive number

This word is updated by the microcode to reflect the drive that is currently selected. The software can force a new drive select by storing 100000B + drive number into this cell.

KBLK+2 -Cylinder Position-

This word is updated by the microcode to indicate the cylinder address last issued to the disk drive. The cylinder address of any successive disk transfer is compared to this value to determine whether a seek command should be issued to the drive. Unnecessary seek commands will cause no problems but will take a full sector time to be completed. The software can force a seek to take place by storing -1 into this cell.

KBLK+3 -Sector Status-

This word is updated by the microcode to indicate the disk status at the last sector pulse. Note that this word in NOT updated when the microcode is busy processing a disk command block.

Bit 0 -Seek Incomplete-

Indicates that the disk drive has not correctly positioned the heads within the last 700ms. A RE-ZERO command must be issued to the drive in order to clear this error.

Bit 1 -Head Overflow-

Indicates that the head address given to the disk drive is invalid (i.e. greater than 4).

Bit 2 -Device Check-

One of the following errors occurred.

a) Head select or Cylinder select or Write commands and disk not ready
b) An illegal cylinder address.
c) Offset active and cylinder select command.
d) Read-Only and Write.
e) Certain errors during writing, such as more than one head selected, no transitions of encoded data or heads more than 80 micro-inches off cylinder.

A RE-ZERO command may be required to clear this error.

Bit 3 -Not Selected-

The selected drive is in "off-line" test mode or the selected drive is not powered up.

Bit 4 -Not On-Line-

The drive is in test mode or the heads are not loaded.

Bit 5 -Not Ready-
There is a cylinder seek in progress or the heads are not loaded.

Bit 6 -Sector Overflow-
The controller detected that a write command was active when the next sector pulse occurred. The controller will inhibit writing as soon as this error is detected. This error implies either a hardware malfunction or a discrepancy between the format of the drive and the format the program thinks the drive has.

Bit 7 -Output Late-
The 16 word output buffer within the disk controller became empty while either a read or a write command was in progress. The controller will inhibit any writing until a Device Check Reset command is issued.

Bit 8 -Input Late-
The 16 word input buffer within the disk controller became full. This error will cause words read into memory to be left shifted by the number of words lost, and the error correcting code to be non-zero.

Bit 9 -Compare Error-
The data read during a "Read and Compare" operation did not match the data read off the disk. The controller will inhibit any writing for the remainder of the sector if this error is detected.

bit 10 -Read Only-
The "Read-Only" switch on the disk drive in on.

Bit 11 -Offset-
The cylinder position is currently offset. This is a mode used for recovery of bad data.

Bit 12 -
Bit 15 -Sector Count-
A value from 0 to count-1, where count is the number of sectors implemented in the disk drive. This value, you might have noticed, restricts the number of sectors to 16 or less. The value returned here is the sector count for the *next* sector on the disk.

KBLK+4 -Command Abort Errors-
This word is set by the microcode if, while processing a command, it finds some extraordinary error. In this circumstance, the status word in the DCB will not be written (though it may have been written in a previous DCB), but the reason for aborting will be reported here.

Bit 0 -
Bit 3 -not used-

Bit 5 -Seek Incomplete-
The last DCB was aborted because the seek was not completed (within 64 sector times).

Bit 6 -

Bit 9 -not used-

Bit 10 -Output Late-
The command chain was aborted because an Output Late error occurred in some previous command. Because commands as well as data are sent to the controller through the output data path, an Output Late command may cause the controller hardware and microcode to get out of sync, so further processing is aborted to prevent issuing illegal commands to the controller. This condition must be reset by issuing a Device Check Reset command.

Bit 11 -Invalid Seal-
The last DCB was aborted because word DCB+4 was not equal to 122645 octal.

Bit 12 -
Bit 13 -not used-

Bit 14 -Aborted-
The last DCB was aborted because of one of the errors mentioned here.

Bit 15 -Invalid Sector-
The last DCB was aborted because the hardware sector counter never equaled the specified sector number (within 64 sector times).

DCB -Cylinder Select-
This word will cause the disk drive to position its heads over the indicated cylinder if the cylinder address is different than the previously selected cylinder. Cylinder positioning causes the disk drive to go non-ready for 6 ms. cylinder-to-cylinder and 55 ms. full seeks. (Typical seek times appear to be around 3 ms. cylinder-to-cylinder and 50 ms. full seek)

Bit 0 -
Bit 3 -all zeros-

Bit 4 -
Bit 15 -Cylinder Select-
This field may take on any value, but be sure that it represents a valid address for the disk drive being used. The microcode will update KBLK+2 with this value.

DCB+1 -Head and Sector Select-
This will select the particular head (or surface) for the next command as well as the sector to be used in the transfer.

Bit 0 -Off Track-
This bit may be activated during a read in order to attempt to recover bad data. When activated, this bit will cause the cylinder positioning mechanism to move 80 micro-inches off track.

Bit 1 -Direction-
This bit determines the direction of off track positioning if bit 0 is set.

Bit 2 -

Bit 4 -not used-

Bit 5 -

Bit 7 -Head Select-
The actual head specification.

Bit 8 -
Bit 11 -not used-

Bit 12 -

Bit 15 -Sector Number-
The actual sector specification. Be sure that you specify a valid sector number. There is no test in the microcode for invalid sector numbers, and the microcode will loop forever looking for an invalid sector! ('Life is hard!').

DCB+2 -Drive Select-
This will specify the disk drive to be involved in the upcoming command.

Bit 0 -
Bit 11 -must be all zeros-

Bit 12 -
Bit 15 -Drive number-
This value is used by the disk controller for selecting the appropriate unit. The microcode will update KBLK+1 with this value.

DCB+3 -Next Command Pointer-
This word is read at the end of a disk command, and its contents are placed in KBLK.

DCB+4 -Disk Command Seal-
This word is tested to see that it equals 122645 octal. If this test fails, then KBLK is set to zero and processing is terminated. This word is overwritten with status information during disk processing so that a DCB table cannot be used twice by the controller without program intervention.

DCB+5 -R/W Command-
This word is first checked for zero. If so, then the sector transfer processing is complete, and the microcode will either go on to the next command or terminate. If the word is non zero, the microcode will then check to see that the cylinder positioning is not active, and wait if it is.

Next the sector count is fetched from DCB+1, and the microcode enters a loop waiting for the sector count in KBLK+3 to equal the desired sector. Once this is found, a "wait for next sector" command is issued and then the block transfer commands are sent to the output FIFO for processing.

If the first command in a DCB is nonzero but has neither the Read nor the Write bit set, the microcode will reset the controller, clearing the Sector Overflow, Output Late, and Compare Error conditions (which inhibit writing on the disk). This is typically done using a Device

Check Reset or Re-Zero command, and should be done only after waiting for the disk to be idle.

The control bits of the read/write command are as follows:

Bit 0 -
Bit 3 -not used-

Bit 4 -Check Data-
This bit is examined during a read command to see whether incoming data should be compared against that data which already exists in the Alto memory. In this mode, the first n non-zero words from the Alto memory are passed to the disk controller for comparison during reading, where n is at least 2, and not greater that the word count for that block. The disk controller will perform a normal read (placing all but the first 2 data words read into Alto memory), and in addition it will set an error bit if the check words are different. This check bit will inhibit any further write operations within the sector.

Bit 5 -not used-

Bit 6 -Strobe late-
This is used in conjunction with read in an attempt to recover bad data.

Bit 7 -Strobe Early-
Same as bit 6. I don't know what happens if both bits are set.

Bit 8 -Write-
Indicates that the transfer is to be a write.

Bit 9 -Read-
Indicates that the transfer is to be a read.

Bit 10 -not used-
Bit 11 -Head Address Reset-
Please don't use this bit, since it will force head 0 to be selected.

Bit 12 -Device Check Reset-
This bit is used to clear any error stored by the device check circuitry. This bit is activated by the microcode and should not be set here except to recover from a Device Check, Compare Error, Output Late, or Sector Overflow. The read, write, and check bits should not be activated when this bit is set.

Bit 13 -Head Select-
This bit causes the head select electronics to activate the selected head. This bit should always be set for a read or a write command.

Bit 14 -Re-Zero-
This is a special control function which is used to cause the disk drive to completely retract its cylinder positioning arms and then reposition them on cylinder 0. This bit should only be activated when a head positioning error has been detected. The read,

write, and check bits should not be activated when this bit is set.

Bit 15 -Advance Head Address-
Please don't use this bit either, since it will increment the head address selected by 1.

DCB+6 -Word Count-
This word specifies the number of words in the block to be read or written. This number does not include the two words of check sum at the end of each block.

DCB+7 -Memory block Pointer-
This is a pointer to the appropriate memory block in Alto memory.

DCB+8 -
DCB+9 -Error Correcting Code-
These two words are updated at the end of a read with the contents of the error code shift registers. If no read error has occurred then both these words will be identicaly zero. If they are non-zero. then an error has occurred. and error correction may be applied. The error recovery process runs as a BCPL procedure. and a small microcode routine. which are capable of determining the error bits in less than 5 ms.

DCB+10 -Status-
This word contains the disk status at the end of processing the current block.

Bit 0 -
Bit 10 -Disk & Controller errors-
The status bits in this field are the same as those described for KBLK+3.

Bit 11 -ECC error-
Indicates that one of the two ECC words was found to be non-zero.

Bit 12 -
Bit 15 -constant-
This field is set to 1 by the microcode. Note that this means a 1 is returned in the entire status word for a transfer with no errors.

5. Format Specifications

Various delays must be provided at the beginning of each sector block in order to allow for electrical and mechanical tolerances within the disk drive. For the purpose of defining a new disk format. one simply needs a summary of "words lost" for each block.

1) total words per disk revolution =	10.080
2) words lost for the 1st block =	32
3) words lost for successive blocks =	14
4) required gap at end of sector =	14

The number given for "words lost" for each block includes: 2 words of error detection and correction (32 bits of ECC code) which are always added at the end of the data written. 1 word of

trailing zeros (to assure that all data is sent before the write electronics is turned off), 1 word for the disk controller to execute the read/write command, and the number of delay words allotted by the disk controller as required by the disk drive for mechanical and electronic delays.

Using these numbers on the standard Alto disk format yields the following numbers. The number of words available per sector is 10,080/9 = 1120. The total words lost for disk formatting is 32 for the first block, 28 for the second and third block, and 14 words at the end of the sector, for a total of 74. Subtracting 74 from 1120 gives us 1046 words remaining for the three fields. This will give us 2 words for Header, 10 words for Label, and 1024 words for Data, with 10 words unused.

6. Error Detection and Correction

The following section describes (as best I can) the provisions implemented for error correction for the Trident disk drives. This capability is done as a mixture of disk controller hardware (for ECC generation and checking) and system-software/microcode (for error recovery).

The error detection and correction scheme implemented in the disk controller is a compromise of capability, speed, and cost. The number of check bits generated is 32 (2 words are easy to keep track of), and the total controller chip count has been held down to fit along with the Ethernet controller, on a single Dorado printed circuit board. The basic capabilities and restrictions are summarized below.

1) Correction of *single* error burst of length not to exceed eleven data bits. (Example: for the data "0001100101", the data "0000101101" contains a single burst error of length 4.)

2) Capable of correcting record lengths of up to 2684 words. The error detection and correction code implemented will detect errors in arbitrary length records, but not enough information is generated for error correction if the sector length exceeds 2684 words.

3) Simple error detection. 2 words are returned by the hardware, which if both words are zero, indicates a successful read.

4) Error correction in less than one revolution of the disk drive. The error correction procedure is well suited for implementation in a mixture of BCPL and microcode. The bulk of the processing, which is in microcode, will take a maximum of 4.5 ms. worth of microinstructions. This is sufficiently fast to allow for reading a sector, finding that an error occurred, attempting to correct it, and if not correctable, initiating a new read of the same sector on the next revolution.

5) Not all uncorrectable errors will be detected as such. The probability of an uncorrectable error being generated is exceedingly small. It requires two bad spots on the disk surface within one sector (the pack is bad - throw it out!), an electronic error in a sector with a bad spot, or two electronic errors within one sector. Given that such an error has occured, it can, with a probability of say 20 percent, result in an error pattern and displacement which is seemingly valid. This will result in leaving

the error bits un-corrected and, in addition, changing some bits which were in fact correct. This means that for high data security, a check code should be generated and imbedded as part of the data file before writing on the disk.

The following is a more detailed description of the error correcting code used, and the procedure used for data recovery.

The error correcting code (ECC) generated is referred to as a Fire Code (see *Error-Correcting Codes* by Peterson), and is capable of correcting any single *burst error* of up to 11 bits in length (that is a scattering of error bits within the bit stream, all of which fit within an 11 bit span). The code calls for dividing the outgoing data stream by a polynomial of the form:

$$P(X) = P_1(X)(X^m + 1)$$

Where $P_1(X)$ is an irreducible polynomial of degree n (n = burst length) and m is $\geq 2*n - 1$. For this particular application the polynomials chosen are:

$$P(X) = (X^{11} + X^2 + 1)(X^{21} + 1)$$

During a write operation, the two polynomials are multiplied together and implemented by hardware in the form:

$$P(X) = X^{32} + X^{23} + x^{21} + X^{11} + X^2 + 1$$

The data stream is premultiplied by X^{32} to make room for the 2 word ECC and then reduced modulo P(X). This is accomplished by the normal feedback shift register technique with the difference that to perform premultiplication, the output of the register is exclusive-or'd with the incoming data and then fed back. After all data bits have been shifted out, the contents of the ECC shift registers are appended to the disk block.

During the read operation, the feedback shift register is reconfigured such that the two original polynomials are implemented separately. The incoming data stream, including the 2 appended words of ECC, is independently reduced modulo $P_0(X)$ and $P_1(X)$, where

$$P_0(X) = X^{21} + 1$$

$$P_1(X) = X^{11} + X^2 + 1$$

After reading in all words off the disk, the contents of the two polynomial shift registers are read into the Alto. If the data is recovered without error, then reducing it modulo $P_0(X)$ and $P_1(X)$ results in the registers containing all zeros.

If the data contains an error, then the two registers will be non-zero. If one but not both registers is non-zero, then the error is non-recoverable.

Given that one finds an error, then a procedure is undertaken which determines the pattern of bits which are in error, and the displacement of this pattern from the end of the record. I am simply going to present the magic equation to be solved, and some magic constants to be used for solving this equation. Much of the polynomial implementation and the equations, which use the "Chinese

Remainder Theorem" are discussed in technical reports from CALCOMP (Calcomp Technical Report TR-1035-04, by Wesley Gee and David George) and XEROX (Xerox XDS preliminary report "Error Correction Code for the R.M. Subsystem, by Greg Tsilikas, March 28, 1972.).

The basic equation is:

$$D = Q*LCM - (A_0*M_0*S_0 + A_1*M_1*S_1)$$

Where:

D = displacement from the end of the record.
Q = smallest integer to make D positive
A_i = a constant such that $A_i*M_i = 1$ modulus E_i
E_i = modulus of the polynomial
M_i = LCM/E_i
LCM = least common multiple of E_0 and E_1
S_i = number of shift operations to the appropriate polynomial remainders as described below.

The values of E_0 and E_1 were found by programing the procedure outlined in the CALCOMP report, and yielded the following result:

$$E_0 = 21 \qquad E_1 = 2047$$

The least common multiple (LCM) of E_0 and E_1 is simply the product of E_0 and E_1 since the two numbers have no factors in common. Thus the LCM, which is also the record length which can be corrected, is 42,987 bits, or 2684 words.

Knowing LCM and E_0 and E_1, the values of M_0 and M_1 are easily found to be

$$M_0 = 2047 \qquad M_1 = 21$$

The values of A_0 and A_1 are next determined using a trial and error approach that I put in a small program. The results can easily be confirmed, and are given below:

$$A_0 = 19 \qquad A_1 = 195$$

All of the above values derived so far are constants determined for the particular polynomials chosen. The values of S_0 and S_1 are determined in the software from the error patterns returned at the end of a disk transfer.

S_0 is first determined through a BCPL procedure by the following steps.

1) The remainder from dividing the input data by $X^{21} + 1$ is found in the first ECC word, bits 11 through 15, and the second ECC word, such that first word bit 11 is the most significant bit and second word bit 15 is the least significant bit.

2) First test the remainder for zero, and if so quit since the error is non-recoverable.

3) Test the low order 10 bits for all zeros, and if not then perform a left circular shift on the 21 bits. When the low order 10 bits are all zeros, the error pattern is in the upper 11

bits of the word, and S_0 is the number of times the circular shift was performed.

4) If the low order 10 bits don't become all zeros within 21 shifts (1 full cycle) an uncorrectable error has occurred.

S_1 is then determined through a microcode procedure by the following steps.

1) The remainder from dividing the input data by $X^{11} + X^2 + 1$ is found in the second ECC word, bits 0 through 10.

2) First test the remainder for zero, and if so quit since the error is non-recoverable.

3) Test this number to see if it is equal to the error pattern determined in step 3 of S_0, and if not reduce this number modulo $X^{11} + X^2 + 1$ (left shift and XOR feedback). When the contents of this word equals the error pattern (it is guaranteed to happen with 0 to 2047 shifts), the value of S_1 is determined as the number of shifts performed (In the hardware implementation of switching from the write polynomial to the read polynomials, it was easier to implement a polynomial that premultiplied by X^{11}. This means that the remainder returned by the hardware already has had 11 shifts performed. To compensate, when S_1 has been determined by the above procedure, you must add 11 to the value, and subtract 2047 if the result is greater than 2047).

The basic equation for the displacement now looks like

$$D = Q*42.987 - 19*2047*S_0 - 195*21*S_1$$

Where:

$$0 \leq S_0 \leq 21$$
$$0 \leq S_1 \leq 2047$$

Notice that the straight-forward solution to this equation can not be done with single precision arithmetic on the Dorado. In order to avoid double precision arithmetic, I have used the following manipulation of the equation.

$$D = Q*2047*21 - 19*2047*S_0 - 4095*S_1$$
$$D = Q*2047*21 - 19*2047*S_0 - 2*2047*S_1 - S_1$$
$$D' = Q*21 - 19*S_0 - 2*S_1$$

where:

$$0 \leq D' \leq 20$$
$$D = 2047*D' - S_1 \qquad \text{(add 42.987 if } D' = 0\text{)}$$

7. Hardware implementation

Due to the high data transfer rates of the Trident disk (1.65 μs per word), the disk controller implementation includes two independent 16 word "First In - First Out" (FIFO) registers. One FIFO is used to buffer all output information, both control and data, and the other FIFO buffers all input data. A status input instruction has been implemented which is not buffered through the

FIFO's.

The two FIFO's are completely independent of one another, and are, in fact, serviced by two separate microcode tasks. With the standard allocation of task assignments in the Alto, the most suitable tasks available for the new disk are task 17, the highest priority task, for the input FIFO and task 3 for the output FIFO. With this implementation, a disk read can be performed without concern over the microcode activity of other tasks. The output task, while being of low priority, will have enough room in the FIFO to store all commands for reading the sector. The input task, which will have all the activity, will have the highest priority. During this disk activity, the standard Alto disk must be idle for its own good, while the display will still function, but with possible break-up of the display during the transfer. During a disk write, all task activity above the disk (i.e. everything but the emulator) must be minimized in order to guarantee adequate service to the disk. If the output FIFO is allowed to become empty, the "write late" flag will be set and the write command to the disk will be inhibited for the remainder of the transfer.

The fullness or emptiness of the appropriate FIFO generates a task wake-up request if there is enough room for at least 4 words to be transferred. This is sufficient to allow a double word memory access to take place in a minimum micro-instruction loop of 6 instructions per double word read/write.

All disk formatting delay constants are implemented with a 32 X 8 PROM. An Alto program is available which takes the required delay values from the user and computes the appropriate values to be put in the PROM. These constants do not involve the number of blocks per sector or the number of data words per block - these being determined from values passed to the microcode in the disk command table.

For applications requiring a single disk drive on an Alto, the hardware implementation involves a single Alto PC card which is plugged into a "processor" slot in the Alto. In the situation requiring more than one drive (with a maximum of 8) an additional Trident Multiplexor card is required. This card contains data and clock repeaters, disk select decoding and latch, and a sector count register. (It should be mentioned that the controller will not work unless drive 0 is connected and has AC power turned on; however, it is not necessary that drive 0 be on-line.)

Software and Utilities for Trident Disks: Tfs and Tfu

---

1. Introduction

This document describes the software for operating any of the family of Trident disk drives attached to an Alto using a "Trident controller card" (TRICON) (the software presently deals with the T-80 and T-300 models). Additional information can be found in the document "Trident disk for the Alto" by Roger Bates which is section 9 of this manual.

A "Shugart controller card" also exists, for connecting to Shugart model SA-4004 and SA-4008 disk drives. The Shugart controller is microprogram compatible with the Trident controller, and the Trident software can operate it as well. In this document, all references to Trident disks apply to Shugart disks as well, except where noted otherwise.

The Tfs package and utilities all assume that the disk is to be formatted with 1024 data words per sector. The maximum capacity of each disk is given in the following table.

Disk	Tracks	Heads	Sectors	Total pages	Total words
T-80	815	5	9	36,675	37,555,200
T-300	815	19	9	139,675	142,709,760
SA-4004	202	4	8	6,464	6,619,136
SA-4008	202	8	8	12,928	13,238,272

For all disks except the T-300, it is possible to construct a single Alto-format file system utilizing the full disk capacity. Due to the restriction of virtual disk addresses to 16 bits, a single file system may utilize only about 47 percent of a T-300 disk, and it is necessary to construct multiple file systems in order to make use of the entire disk.

Because of bandwidth limitations, it is unwise to operate the Trident disk while the Alto display is on. Although the Tfs package will save the display state, turn it off, run the disk, and restore the display for every transfer, the user may prefer to turn the display off himself. The Tfs management of the display causes the screen to flash objectionably whenever frequent calls to Tfs are underway.

The present version runs only under Operating System version 16 or newer.

2. Trident File Utility, Tfu

The Tfu utility (Tfu.Run) is used to certify a new Trident pack for operation, to initialize a pack with a virgin file system, and to perform various file copying, deleting, and directory listing operations.

Commands are given to Tfu on the command line: immediately following the word "Tfu" is a subcommand name (only enough characters of a subcommand are needed in order to distinguish it from other subcommands), followed by optional arguments. Several subcommands may appear on one command line, separated by vertical bars. Thus "TFU Drive 1 | Erase" will erase drive 1. There must be a space on each side of the vertical bar.

All information shown on the display by Tfu is also written into file Tfu.log (on the Diablo disk). Certain commands pause and type "Continue?" after each screenful: type "space" to proceed.

In what follows, an "Xfile" argument is a filename, perhaps preceded by a string that specifies which disk is to be used:

DP0:name.extension -- use standard Alto (Diablo) disk
TPn:name.extension -- use Trident drive n (n=0 to 7)
name.extension -- use default disk (Trident)

The "default disk" is always a Trident drive: the identity of the drive is set with the Drive **command.**

TFU DRIVE driveNumber

This command sets the default Trident drive number to use for the remainder of the command line. The default drive is effectively an 'argument' to the CERTIFY, ERASE, DIRECTORY, CONVERT, and BADSPOTS commands. (On a T-300, file systems 0, 1, and 2 are specified as 'TPx', 'TP40x', and 'TP100x', where 'x' is the actual unit number.)

TFU CERTIFY [passes]

This command initializes the headers on a virgin Trident disk pack, then runs the specified number of passes (default 10) over the entire pack, testing it using random data. Any sector exhibiting an uncorrectable ECC error, or correctable ECC errors on two or more separate occasions, is permanently marked unusable in the pack's bad page list. This information will survive across all subsequent normal file system operations (including TFU ERASE), but may be clobbered by the Triex program.

This command should be executed on every new Trident pack before performing any other operations (such as TFU ERASE). 10 passes of TFU CERTIFY are adequate for reasonably thorough testing, though more are recommended for packs to be used in applications requiring high reliability. The running time per pass for TFU CERTIFY is approximately 3 minutes on a Trident T-80, 9 minutes on a T-300, and 1.5 minutes on a Shugart SA-4008.

TFU CERTIFY may be terminated prematurely by striking the space bar to get its attention, then typing 'Q'. Subsequent runs of TFU CERTIFY will not clobber the existing bad page information but rather will append to it. It is recommended (though not necessary) that TFU CERTIFY be executed before each TFU ERASE so as to pick up any new bad spots that may have developed.

TFU CERTIFY ordinarily asks you to confirm wiping out the disk before going ahead and doing so; however, the /N global switch may be used to indicate that no confirmation is necessary.

TFU BADSPOTS

Displays the addresses of all known bad spots on the disk pack mounted on the default drive.

TFU RESETBADSPOTS

Resets the bad spot table of the disk pack mounted on the default drive. (Note that TFU CERTIFY appends to the existing bad spot table.) There should normally be no need to execute this command, but it may be useful, for example, after a disk pack is cleaned, if the known bad spots were caused by dirt.

TFU ERASE [tracks]

This command initializes (or reinitializes) a file system on the pack mounted on the default Trident drive, after asking you to confirm your destructive intentions (overridden by the /N global switch). The tracks argument specifies how many "tracks" of the drive are to be included in the file system; it defaults to the maximum possible. If smaller numbers are used, the initialization is correspondingly faster. In any case, tracks beyond the one specified are available for use outside the confines of the file system. (Note that one "track" is 45 pages; this corresponds to one cylinder on a T-80 and to nothing in particular on other disks.)

The disk pack should previously have been initialized and tested by means of the TFU CERTIFY command.

The DiskDescriptor file is normally located in the middle of the file system so as to minimize average head movement between DiskDescriptor and file pages. However, this does limit the maximum size contiguous file that can be created to a little less than half the file system. If you wish to create a contiguous file larger than that, use the /B local switch (i.e., TFU ERASE/B) to force the DiskDescriptor to be located at the beginning of the file system instead.

TFU COPY Xfile ← Xfile
This command copies a file in the direction of the arrow. The destination file may be optionally followed by the switch /C, in which case (provided it is a Trident disk file), the file will be allocated on the disk at consecutive disk addresses. (Note: More precisely, an attempt will be made to perform such an allocation. If the attempt fails, you will sometimes get an error message. The best way to verify that a file is contiguous is to use the "address" command, below.)

TFU CREATEFILE Xfile pages
This command creates a contiguous file named Xfile with length "pages."

TFU DELETE Xfile Xfile ...
This command deletes the given file(s).

TFU RENAME Xfile ← Xfile
This command renames a file.

TFU DIRECTORY [Xfile]
This command lists the directory of the default Trident drive on the file Xfile; if Xfile is omitted, each entry will be shown on the display. A somewhat more verbose listing can be obtained with TFU DIR/V.

TFU ADDRESS Xfile
This command reads the entire file and displays a list (in octal) of virtual disk addresses of the file pages.

TFU CONVERT
An incompatible change in the format of DiskDescriptor was made in the Tfs release of July 24, 1977. The current Tfs software will refuse to access Trident disks written in the old format. The TFU CONVERT command reformats the DiskDescriptor to conform to current conventions (it is a no-op if applied to a disk that has already been converted). Once you have converted all your Trident disks, you should take care to get rid of all programs loaded with the old Tfs, since the old Tfs did NOT check for version compatibility.

TFU EXERCISE passes drive drive drive ...
This command embarks on a lengthy "exercise" procedure; it is repeated 'passes' times (default=10), and uses the disk drives listed after 'passes' (if none are specified, all drives that are on-line are used). It operates by making a series of files (test.001, test.002 etc.) on the disk packs, and performing various copying, deleting, writing and positioning operations. The files are deleted when the exercise finishes. It is not essential that the packs be fully erased initially; the procedure for building test files will try to fill up the disk, just short of overflowing. Each pass of the test takes approximately 20 minutes per T-80, 60 minutes per T-300, and 10 minutes per SA-4008.

One or more of the following global switches may be specified (i.e., a command of the form TFU/switch EXER...):

/W Use a systematic data pattern when writing files, rather than arbitrary garbage.

/C Carefully check the data read from the disk (implies /W). Use of this switch makes the test run considerably slower than normal.

/D Leave the display on during Trident disk transfers. This causes data late errors to occur and thereby exercises the error recovery logic. (It also slows down the test by at least a factor of 10.)

/E Turn the Ethernet on during Trident disk transfers, with results similar to /D.

3. File structure on the Trident disk

The file structure built on the Trident disk by Tfs (Trident File System) is as exact a copy of the Alto file structure built by Bfs (Basic File System) as is possible. Certain exceptions are present due to hardware and microcode differences.

3.1. Disk Format

The Trident or Shugart disk drives are set up to run with the following parameters:

Disk	Cylinders	Heads	Sectors
T-80	815	5	9
T-300	815	19	9
SA-4004	202	4	8
SA-4008	202	8	8

TFU CERTIFY will format each sector of the disk in the standard Tfs format:

header words per sector: 2
label words per sector: 10
data words per sector: 1024

Thus, for example, a T-80 disk will have 9*5*815 = 36.675 sectors = 37.555.200 words. Sector 0 will not be used by Tfs. All but sector 0 will be available to the file system.

Ordinarily, Tfs utilizes only the first 383 cylinders (= 65.493 sectors = 67.064.032 words) of a T-300 disk. This is the largest integral number of cylinders that can be addressed using a 16-bit virtual disk address. The 16-bit virtual address limitation is deeply embedded in all existing higher-level Alto file system software, so changing the Tfs interface to permit a larger virtual address space would be impractical.

Instead, Tfs permits one to obtain another, entirely independent disk object for referencing the second 383 cylinders of the same T-300, thereby permitting a separate, self-contained file system to be constructed. A third file system may also be constructed, but it contains only 49 cylinders (= 8379 pages, only 6 percent of the disk's total capacity), so doing so is probably not worthwhile.

3.2. Disk Header and Label

On the Trident, a real disk address requires two words to express, rather than the single word on the Diablo 31. Also, microcode considerations gave rise to a reordering of the entries in the Label. The result is that both the header and label formats are different for the Trident. The Trident format follows.

disk header contains:
    track word
    head byte
    sector byte

disk label contains:
    fileid word
    packID word
    numChars word
    pageNumber word
    previous @DH
    next @DH

3.3. Disk Descriptor

Every valid Tfs disk has on it two files which must contain the state information necessary to maintain the integrity of the file system. The Tfs system directory, "SysDir.", is identical in format and purpose with its Bfs counterpart. However the Tfs disk descriptor file, "DiskDescriptor.", while identical in purpose, is formatted differently to allow easy manipulation of the bit table (which, for the Trident, has to be paged in and out of memory). This difference in format should not be evident to even low-level Trident users but is mentioned here for completeness.

3.4. Bad Page Table

Tfs and Tfu observe the standard Alto file system convention of recording -2's in the labels of all known bad pages. However, if this were the only location of such information, "erasing" a disk (to create a virgin file system) would require two passes over the entire disk: one to collect the addresses of all known bad pages and one to mark all remaining pages deleted. This would require an excessive amount of time, particularly on a T-300.

A duplicate table of known bad pages is therefore recorded on physical page zero (= cylinder 0, head 0, sector 0) of the disk. This page is not available to the file system for other reasons having to do with end-of-file detection. Note that the entries are REAL disk addresses and can therefore refer to any page on the disk regardless of whether or not such a page is accessible through the file system. (A T-300 has only one bad page table, even if it contains several file systems.)

The TFU CERTIFY command is responsible for testing the pack and building the bad page table. The TFU ERASE procedure is careful not to clobber this information but rather to propagate it to the other places where it is needed (namely, the disk bit table and the labels of the bad pages themselves). As a result, the bad page information, once initialized, will survive across all normal operations on the disk, including "erase" operations.

There does not presently exist any facility for manually appending to this list when new bad pages are discovered. Experience to date with the Trident disks (which provide correction for error bursts of up to 11 bits in length) has shown that such a facility is probably not needed. Thorough testing of disks (using TFU CERTIFY) is recommended before putting them into regular use, however.

Inter-Office Memorandum

To	IFS Users	Date	September 6, 1980
From	Ed Taft and David Boggs	Location	Palo Alto
Subject	How to Use IFS (version 1.27)	Organization	PARC/CSL

XEROX

Filed on: [Maxc1]<IFS>HowToUse.bravo, .press

This memo describes how to use the IFS (Interim File System) servers. This is the complete user-level documentation. The Alto User's Handbook has some introductory material and summarizes commonly-used procedures, so new users are advised to look there first.

The names of some of the IFSs presently operating, and the electronic mail addresses of persons to contact concerning accounts, are as follows:

IFS name	*Organization*	*Administrator*
Ivy, Phylum	PARC (Palo Alto)	RWeaver.PA
Iris, Igor, Idun	SDD (Palo Alto)	SDSupport.PA
Isis, Sun, Wind	SDD (El Segundo)	SDSupport.PA
Ibis, Ibid	OPD (Palo Alto)	SDSupport.PA
Oly	OPD (El Segundo)	Nikora.ES
XEOS	EOS (Pasadena)	DonWinter.EOS
ADL	ADL (El Segundo)	RHunt.PA
Erie	WRC (Webster)	Axelrod.WBST
Cactus	Dallas	Yost.DLOS
Calypso	Leesburg	
Eagle	Corporate headquarters (Stamford)	
Aklak	XRCC (Toronto)	

These names are the ones used to identify specific IFSs to the FTP, Chat and CopyDisk subsystems. Information in this memo applies to all IFSs except where otherwise noted.

This edition describes IFS version 1.27. The only user-visible change since the previous version is addition of a 'Printed-by' subcommand to the 'Print' command.

How to access IFS

At present, the file services provided by IFS are limited to a fairly basic set. The normal mode of access from Altos is through FTP. The basic operations (Store, Retrieve, List, Delete, and Rename) are invoked through FTP in precisely the same manner as when accessing Maxc. The only difference is that you request FTP to open a connection to some IFS (by specifying its name) rather than Maxc.

You should consult the FTP documentation in the Alto User's Handbook, the Alto Subsystems manual, or [Maxc1]<AltoDocs>FTP.tty, for general information on the use of FTP. IFS can also be reached from Maxc by means of the PUPFTP subsystem.

File naming conventions on IFS are a mixture of Maxc and Alto conventions. The general form of an IFS file name is:

<directory>name!version

All printing characters except '*' are legal in the name. The complete file name may be up to 99 characters long (longer than either Maxc or Alto permit).

All IFS files have version numbers (in the range 1 to 65535) which are defaulted in the usual way, as follows:

Retrieve	highest existing version
Store	next higher version
Delete	lowest existing version
List	all versions

Versions other than the default one may be referred to explicitly (by specifying the version number) or by the notations '!L' (lowest existing version), '!H' (highest existing version), or '!N' (next higher version).

There is presently no facility for automatic deletion of non-current versions, but such a feature may be implemented eventually.

'*' expansion is supported during Retrieve, List, and Delete commands. The expansion is similar to that provided by the Alto Executive; that is, each '*' matches zero or more real characters in a file name.

You may find it convenient to organize your files into *sub-directories* by giving them names such as '<Taft>Memos>HowToUse.Bravo'. Then all files belonging to a particular sub-directory may be accessed by a specification such as '<Taft>Memos>*', and you may direct your attention to a particular sub-directory by establishing a default such as 'Directory Taft>Memos'. The system does not presently attach any important semantic significance to the sub-directory notation, but this may change eventually.

Access via Chat

The current definition of the File Transfer Protocol (the means by which FTP communicates with a file server) limits itself to the basic set of operations mentioned previously. It lacks the means for expressing a number of other essential operations. Improved file access protocols are a topic of current research.

In the meantime, rather than attempting to extend FTP, we have provided an Executive in IFS which you can access by means of Chat (or the bottom *Telnet* window in FTP). This Executive is patterned after the one in Maxc, but has a very limited command repertoire.

Typein and editing conventions are the ones familiar to most users. BS and CTRL-A erase the preceding character. CTRL-W deletes a word, and DEL deletes an entire command or sub-command. Deleted characters are not actually erased from the Alto screen because Chat does not provide such a capability. Most commands must be terminated by RETURN. CTRL-C may be used to abort any command. If you are using any sort of display terminal, typeout will stop at the end of every page (as on Maxc) and IFS will wait for you to type any character before continuing. If you type ahead, this feature is disabled.

The current commands of interest to most users are the following:

@ Login (user) *user-name* (password) *password*

Logs you into IFS. This is necessary before issuing most other commands. Ordinarly, Chat will do this for you automatically.

@ Logout
@ Quit

Logs you out and closes the connection.

@ Connect (to directory) *directory-name* (password) *password*

Sets your default directory to be *directory-name*, and gives you owner-like access to it. The *password* may be omitted if *directory-name* is your own directory or one to which you have connect privileges.

@ Directory (default) *directory-name*

Sets your default directory to be *directory-name*, but without changing your access rights (and therefore without requiring a password). All subsequent commands dealing with files will behave as if '<*directory-name*>' appeared at the beginning of each file name argument that doesn't name a directory explicitly (i.e., that doesn't begin with '<'). *Directory-name* may include sub-directories (e.g., '<Jones>Memos>').

When you issue the 'Directory' command, IFS first displays your current default directory. You may either edit this field (by first backspacing at least one character) or replace it simply by typing the replacement. If you erase the entire field (with CTRL-W), the default directory reverts to your current connected directory.

If the first character of *directory-name* is '>', IFS prefixes the name of your current connected directory. That is, if you are currently connected to directory Jones, the command 'Directory >Memos>' is equivalent to the command 'Directory <Jones>Memos>'. Also, the outermost '<' and '>' are optional. Note that the foregoing descriptions also apply to the Directory command in the FTP server.

@ DskStat

Prints the number of used pages and the maximum allowed in the connected directory, followed by the number of free pages in the system. One IFS page is 1024 words or 2048 characters, which is equivalent to four Alto pages or approximately one Maxc page.

@ List (files) *file-designators*

Lists the names of all files matching *file-designators*, which is a list of up to 10 file names (separated by spaces), any of which may contain '*'s to denote multiple files. The files matching each *file-designator* are listed in alphabetical order on the basis of the entire file name (including directories and sub-directories, if any). To save space, directory and sub-directory names are printed only when they change, above the list of files to which they apply.

If you terminate the last *file-designator* with a comma followed by RETURN (rather than just RETURN), IFS enters a sub-command mode in which you may specify additional information to be printed about each file:

@@ Type	file type and byte size
@@ Size	size in pages
@@ Length	length in bytes
@@ Creation	date of file creation
@@ Write	date of last write
@@ Read	date of last read
@@ Backup	date of last backup
@@ Times	times as well as dates
@@ Author	creator of file
@@ Protection	file protection
@@ Verbose	same as Type Size Write Read Author

@@ Everything

Sub-command mode is terminated when you type just RETURN in response to the '@@' prompt. The columns of printout will be aligned properly only if you are running Chat with a fixed-pitch font such as Gacha12 or Gacha10.

@ Delete *file-designator*

Deletes all files matching *file-designators*, which is a list of up to 10 file names (separated by spaces), any of which may contain '*'s to denote multiple files. The version number defaults to the lowest existing version; to delete all versions, you must end each *file-designator* with '!*'. IFS prints out each file name, followed by '[Confirm]'. You should respond with 'Y' or RETURN to delete the file, or with 'N' or DEL to leave it alone.

If you terminate the last *file-designator* with a comma followed by RETURN, IFS enters a sub-command mode in which you may request the following additional actions:

@@ Confirm (all deletes automatically)

IFS will not ask you to confirm deleting each file but will just go ahead and do it.

@@ Keep (# of versions) *number*

IFS will retain the *number* most recent versions of each file and delete all remaining versions. That is, to delete all but the most recent version of each file, specify 'Keep 1'.

On IFS (unlike Maxc), files are deleted immediately; there is no Undelete command. To delete a file, you must have write access to it.

@ Rename *existing-filename* (to be) *new-filename*

Changes the name of *existing-filename* to be *new-filename*. It is permissible to change any part of the file name, so it is possible to move a file from one directory or subdirectory to another by renaming it. The Rename operation requires that you have write access to the file and create access to the directory into which the file is being renamed.

It is permissible to rename a file to itself in order to change its capitalization. Note that a new version of a file always inherits the capitalization of the previous version; renaming a file to itself (i.e., with the same version number) is the only way to defeat this.

@ Print (files) *file-designator*
@ Press (files) *file-designator*

Requests that all Press files matching *file-designator* be sent to your default printing server ('Print' and 'Press' are synonyms). *File-designator* is a list of up to 10 file names (separated by spaces), any of which may contain '*'s to denote multiple files. IFS prints out the name of each file followed by '[Confirm]'; you should respond with 'Y' or RETURN to print the file, or with 'N' or DEL to skip over it.

If you terminate the last *file-designator* with a comma followed by RETURN, IFS enters a sub-command mode in which you may specify the following parameters:

@@ Copies *number*

Specifies the number of copies of each Press document to print.

@@ Server *server-name*

Specifies the name of the printing server to which the Press files are to be transmitted. This may be either a registered name or an internetwork address of the form '*net# host#*' (don't leave off the trailing '#').

@@ Printed-by *name*

Causes *name* to appear in the 'Printed-by' field on the cover page of the printed output. (Ordinarily, your user name is printed.)

You terminate sub-command mode by typing RETURN in response to the '@@' prompt. In the absence of any sub-commands, IFS will cause one copy of each Press file to be printed on your default printing server. You may establish or change your default printing server by means of a sub-command of the 'Change Directory-Parameters' command, as follows:

@ Change Directory-Parameters (of directory) *directory*
@@ Printing-Server *host-name*

where *directory* is the name of your directory, i.e., your user name. If you have not established your default printing server, IFS will require you to issue a 'Server' sub-command every time you request printing.

Actual transmission of the Press files to the printing server is performed by a background process, so you need not remain connected to IFS while the printing is taking place. If the printing server is down at the time, IFS will queue the files for later delivery. If the Press files cannot be delivered within eight hours, however, the printing request is discarded without a trace.

Printing request may be examined and canceled with the following commands:

@ Show Printing-requests

displays all printing requests you have issued that have not been completed.

@ Cancel (printing requests)

displays all outstanding printing requests you have issued, and for each one asks you whether or not you wish to cancel it (answer 'Y' or 'N').

Note that *only* Press-format files can be printed: IFS checks that every file is a Press file and will refuse to print any file that is not.

@ Change Password (of directory) *directory-name* (old password) *password* (new password) *password*

Changes the password of the specified directory, which must be either your own or the one to which you are presently connected. (Contrary to normal practice, the new password does print out as you type it: this is so that if you make a typing mistake you will be able to see it.)

@ Change Protection
@ Change Directory-Parameters
@ Show Directory-Parameters
@ Change Group-Membership
@ Show Group-Membership

See the section describing protections (below).

@ Systat

Shows who is presently using IFS, what service they are accessing (FTP, Telnet, CopyDisk, or Mail), and the name or inter-network address of the machine they are coming from.

@ DayTime

Displays the current date and time.

@ Statistics

Prints out various operating statistics that are generally of interest only to IFS administrators.

Protections

IFS has a reasonably flexible file protection mechanism, but with a somewhat primitive user interface at present. Fortunately, the default protections are the ones appropriate for most users, so you will probably not need to deal explicitly with protections very often.

Your access to files and directories is permitted or denied on the basis of your membership in *user groups*. Every user is a member of a user group called 'World'. You are a member of another user group called 'Owner' with respect to files in your own directory, and temporarily to files in any other directory to which you connect (using the Connect command in FTP or Chat). Additionally, you may be a member of one or more other user groups with numbers in the range 0 to 61. Such numbered user groups generally correspond to specific projects, and are assigned independently within each IFS by that IFS's administrator.

A *file protection* specifies, for each individual file, what types of access are permitted to which groups. There are three types of file access: *read*, *write*, and *append*. If you have read access to a file, you are permitted to read (i.e., retrieve) its contents. Similarly, write access permits you to overwrite, delete, or rename the file, and append access permits you to append to an existing file, even if you don't have write access. IFS does not yet provide facilities for appending to files, but such a capability may be implemented in the future.

The standard default file protection permits read, write, and append access to the Owner and read access to the World. Hence if the file is in your own directory or the directory to which you are connected, you may do anything to it; otherwise you may only read it. But, for example, if the file protection also permits write access by group 3, and you are a member of group 3, then you may overwrite (or delete or rename) the file, even if it is not in your directory or the directory to which you are connected. Note that the read, write, and append access types are independent. It is therefore possible, though perhaps not particularly useful, for a file protection to permit writing but prohibit reading by some user group.

In addition to the protection associated with each file, there are some protections associated with a directory as a whole. The first is the *default file protection* for files in that directory. When a file is created, its protection is assigned in one of two ways. If there is an existing version of the same file, then the new file inherits its protection. More precisely, when version *n* of a file is created, it inherits the protection of the highest-numbered existing version less than *n*, if there is one. Otherwise, the protection assigned is the default file protection of the directory in which the file is being created.

There are two additional types of access to the directory: *create* and *connect*. If you have create access to a directory, then you are permitted to create new files in that directory. If you have connect access to a directory, you are permitted to connect to that directory without giving its password. As with file protections, these types of access are granted or denied individually to Owner, World, and each numbered user group. The standard directory protection permits create and connect access only to the owner.

Each files-only directory has an *owner*. The owner of a files-only directory is permitted to connect to that directory without giving a password, regardless of the connect protection of the directory. This feature avoids the need to define one-member user groups in order to grant owner access to files-only directories

managed by a single person.

The Chat Executive contains several commands by means of which you may manipulate protections of files and directories.

@ Change Protection (of files) *file-designators*
@@ *sub-commands*

Changes the protection of all files matching *file-designators*, which is a list of up to 10 file names (separated by spaces), any of which may contain '*'s to denote multiple files. You specify the changes to be made by means of one or more of the following sub-commands:

@@ Read (access permitted to) *groups*
@@ Write (access permitted to) *groups*
@@ Append (access permitted to) *groups*

where *groups* is a list of up to 10 instances of 'Owner', 'World', or group numbers (separated by spaces) to which the specific access type is to be granted. 'None' may be used in place of *groups* to specify that access is to be denied to all groups. You may precede a sub-command by the word 'No' to specify individual groups to which access is to be denied. The changes take effect when you type RETURN immediately after the '@@' prompt.

Normally, the changes that you specify by means of these sub-commands are *incremental*. That is, the only access/group combinations that are changed are the ones you mention explicitly, while all the remaining ones are unchanged. However, there is an additional sub-command,

@@ Reset (all existing access)

that denies all types of access to all groups. In this case, the entire file protection is changed to permit only those access/group combinations that you enable explicitly.

You may change the protection of any file to which you presently have write access, and of any file in your own directory or one to which you are connected regardless of its protection. That is, you can change the protection of any file of your own even if its present protection does not permit read, write, or append access by you.

@ List ...

The 'Protection' sub-command to the 'List' command (described previously) displays a file's protection thus:

R: *groups*; W: *groups*; A: *groups*

For example:

R: Owner World; W: Owner 3 19; A: None

@ Change Directory-Parameters (of directory) *directory-name*
@@ *sub-commands*

Changes the information associated with the directory as a whole in the manner specified by the *sub-commands*. The directory must be either your own or one to which you are connected.

You may change the default file protection by means of the 'Read', 'Write', and 'Append' sub-commands in the same manner as in the 'Change Protection' command. Additionally, you may change the create and connect access using the sub-commands:

@@ Create (access permitted to) *groups*
@@ Connect (access permitted to) *groups*

The 'No' prefix may be applied to these as well as to the others.

The 'Reset' sub-command requires an additional keyword to specify what it is that you wish to reset:

@@ Reset Default-File-Protection
@@ Reset Create-Protection
@@ Reset Connect-Protection

You may change your default printing server by means of the sub-command:

@@ Printing-Server *host-name*

The changes are not actually made until you type the confirming RETURN in response to the '@@' prompt.

@ Show Directory-Parameters (of directory) *directory-name*

Displays all information about *directory-name*, and additionally prints some other parameters, such as the disk limit and the owner of a files-only directory, that may be changed only by an IFS administrator. If *directory-name* is your own directory, your user group membership is also shown.

An IFS administrator can change any directory parameters for any user. Additionally, an administrator can assign you to be the *owner* of one or more user groups. If you are the owner of a group, you are permitted to change and examine the membership of that group, using the following commands:

@ Change Group-Membership (of group) *group*
@@ *sub-commands*

The sub-commands are one or more of the following:

@@ Add *user-name*
@@ Remove *user-name*

These cause the specified users to be added to or removed from the group. The sub-commands take effect immediately. You exit sub-command mode by typing RETURN immediately after the '@@' sub-command prompt.

@ Show Group-Membership (of group) *group*

Displays the list of users who are members of the specified group. This command takes a long time to complete, because it has to read the directory parameters of every user in the system.

Mail server

IFS optionally makes available a mail server compatible with the Laurel message system. Each geographical area has a registry of mailboxes for all Alto users in that area: at present, the registries are called PA (Palo Alto), ES (El Segundo), EOS (Pasadena), WBST (Webster), HENR (Henrietta), DLOS (Dallas), and XRCC (Toronto). In the current implementation, each registry corresponds to a single mail server machine that contains all the mailboxes within that registry; that is, the registry names are simply aliases for machines. The PA registry is on Maxcl and the other registries are on local IFSs.

If you are in Palo Alto, you will be assigned a mailbox in the PA registry (i.e., you will be given an account on Maxc1); if you are outside Palo Alto, you will be assigned a mailbox in your own local registry. In any event, your registry must be identified in your Laurel.profile, which should look something like this:

Registry: *registry-name*
Hardcopy: *printer-host-name*
Printed-by: $

To send a message to a user whose mailbox is within your own registry, you need only specify that user's name when you are composing the recipient list in Laurel. However, to send to a user in some registry other than your own, you must specify a recipient name in the complete form

user.registry

For example, if your own registry is ES (El Segundo) and you wish to send a message to Jones, who is also in El Segundo, you need only specify 'Jones' (though it is also correct to say 'Jones.ES'). But if you wish to send a message to Smith in Palo Alto, you must specify 'Smith.PA'.

CopyDisk server

IFS contains a CopyDisk server compatible with the CopyDisk program available from the NetExec. When CopyDisk prompts you for a disk name, you can specify a 'disk' on IFS by typing, for example: '[Ivy]<BasicDisks>NonProg.disk'.

We expect that this server will be primarily used to distribute copies of the basic Alto disks, eliminating the need for physical disk packs which often get mislaid. No doubt other applications will evolve with time. By convention, files in CopyDisk format have extension '.disk'.

CopyDisk files can be quite large. A single Diablo 31 disk takes 1275 IFS pages—more than a typical user's entire disk allocation. CopyDisk does not copy free pages, so that number is the worst case for a completely full disk: none the less, it is easy to generate gigantic files that use up your disk allocation.

File backup

Reliability of file storage is accomplished by two facilities, both of which are now operational. First, we have a Scavenger capable of reconstructing the IFS directory from redundant information kept in the file system. We expect to be able to recover from most file system crashes in this manner, with no loss of user files.

Second, we have an automatic backup system that periodically copies files to a backup disk pack. The backup system runs between 2:00 and 5:00 a.m. every day (users accessing IFS during that time may notice some significant degradation in performance). During each backup run, all files not previously backed up or last backed up more than 30 days ago are copied.

This backup system serves two purposes. First, if the file system fails catastrophically in a way that the Scavenger can't recover from, we will be able to reconstruct the file system from backup, with at most one day's files lost. Second, files accidentally deleted or overwritten by users will usually be recoverable if the loss is noticed within 30 days. (The recovery procedure is not particularly convenient, so please don't depend on it as a regular service.)

Present limitations and future plans

IFS now provides facilities sufficient to make it a useful service. It is unlikely that any further major development will be undertaken. IFS has already far exceeded its intended "interim" specifications, and will ultimately be replaced by better facilities.

A major problem is that of performance of the file system. An IFS is nothing more than an Alto with some large disks connected to it. There is insufficient capacity (particularly main memory) in the IFS Alto to support more than a small number of simultaneous users.

We are presently imposing a relatively small limit (somewhere between 4 and 10) on the number of concurrent connections—FTP, Mail, CopyDisk, and Chat users combined. When this limit is reached, the system will refuse to accept additional service requests. To prevent idle users from tying up these precious slots, the IFS will break connections after a relatively brief period of inactivity.

We would be pleased to receive reasonable suggestions for changes or improvements in the set of facilities provided by IFS. However, please be conscious of the limited manpower available for implementing such improvements.

Acknowledgments

Implementation of IFS would have been impossible without the assistance and cooperation of several individuals who have contributed considerable effort in support of this project. Peter Deutsch provided the Overlay, VMem, and ISF packages and implemented a number of improvements needed by IFS. Ed McCreight made available his B-Tree package, which is used for maintaining user directories, and likewise contributed IFS-related improvements. Bob Sproull and Roger Bates sank considerable energy into the Trident disk hardware, microcode, and software to make it work reliably. And Steve Butterfield initially implemented the Mail facilities and made some important internal improvements.