Inter-Office Memorandum

To	PARC/SDD	Date	November 1, 1977
From	Ed Taft	Location	Palo Alto
Subject	Interim File System Overview (edition 3)	Organization	PARC/CSL

XEROX

Filed on: <IFS>Overview.bravo

This memo is an overview of the Interim File System. It includes information on present status and immediate plans, and contains pointers to other documents covering various aspects of design, implementation, and operation.

Background

In the fall of 1975, it was proposed that we construct an interim Alto/Trident file server to relieve the Maxc storage crunch until such time as the Distributed File System (now known as Juniper) is completed and put into service. This proposal was shelved after consideration of manpower requirements.

A year later, the situation was somewhat changed. The Maxc storage situation was, if anything, worse, and the DFS had proceeded more slowly than anticipated due to other commitments by the participants. In the meantime, however, practically all the software required to build an interim server had already been written.

David Boggs and I therefore began implementing an Interim File System (IFS). It initially provides essentially the same facilities as are now available on Maxc for file storage via FTP, with a few additional operations (e.g., delete, archive) not previously implemented in FTP. The hardware on which this system runs, consisting of an Alto with six Trident disk drives, has been installed in the Maxc machine room. An initial version of the software has been completed and the system is now available for use.

A number of other organizations have installed or made plans to install Interim File Systems of their own, thereby reducing dependence on Parc's central facilities. These include SDD (Palo Alto and El Segundo), EOS, ADL, and ORG (for the Santa Clara branch office).

Overview of the Design

The IFS is easily the largest conglomeration of existing software packages ever to be assembled at Parc. Indeed, this project would be quite impractical were it not for the existence of a large number of Alto BCPL packages made available by certain conscientious and public-spirited individuals. These include:

Peter Deutsch:

Overlays (code swapping)
VMEM (virtual memory)
ISF (random access to files)

Bates, Melvin, and Sproull:

TFS (Trident file system) software and microcode

Ed McCreight:

B-Tree package
Sort package

Boggs and Taft:

Pup package
FTP package

Most of these packages have required some degree of modification to adapt them to IFS (chiefly to make them work in a multi-process environment). The authors of the packages have expended considerable effort in making these modifications. Their contributions are greatly appreciated.

The existing Alto FTP server has been modified to provide the following additional capabilities:

Directories, passwords, protections, etc.

Multiple instances of the server running simultaneously.

New commands such as delete, rename, list directory, etc.

Aside from this, the only major new pieces of software we have needed to write are:

A replacement for the "Dirs" module in the Alto Operating System. The IFS is not using the standard Alto directory structure; rather, all files are entered into one directory organized as a B-Tree. This permits efficient access to a very large number of files (say, 25,000) and provides a straightforward way of implementing user directories and subdirectories.

A server Telnet (Executive) implementing operations for which the File Transfer Protocol has no provisions. These include user account management (creating and destroying directories), operational support such as controlling the backup system, and various miscellaneous commands.

A Scavenger program capable of dealing with a file system consisting of several hundred thousand pages.

A backup system that copies files from the primary file system to a backup disk pack, much as is done on Maxc. Each backup pack is organized as a separate, self-contained Interim File System.

An archive system that archives files onto tape (again, much the same as on Maxc). This part of the project will be deferred until we have had an opportunity to write software for the Alto Magtape system enabling one to access tapes over the Ethernet.

In the process of assembling and interfacing these various pieces of software, we have built up a very flexible and convenient programming environment. This environment has been exported for use in another application (a system for indexing the American Heritage dictionary). We believe it would not take much effort to organize the "IFS environment"

into a self-contained software package for general use.

The major feature of the IFS environment is a relatively complete integration of code overlays, virtual memory, and storage management. The Overlay and VMem packages already mesh in a fairly clean way (the VMem package is used to manage storage for overlays). We have added a virtual memory allocator that permits separate regions of virtual memory to be used in distinctly different ways.

The IFS storage allocator is based on the standard Alto OS allocator and on the VMem package. Two regions of memory are dedicated to standard allocator zones for obtaining small and medium-size blocks (segregating the block sizes reduces fragmentation). Larger blocks (500 words or greater) are obtained by removing whole pages from VMem's buffer pool. When a zone overflows, the allocator recovers by temporarily taking storage from the next larger zone or from the VMem pool. This results in some temporary degradation of performance as opposed to catastrophic failure.

All of memory not devoted to resident code or storage zones is assigned to the VMem buffer pool. In the present system, this is approximately half the Alto memory. This single buffer pool (managed by VMem) is used to satisfy requests for all large blocks of storage, including code overlays, B-Tree pages, and large allocated objects such as stream buffers.

Other Documents

The following IFS-related documents are presently available (all in the <IFS> directory on Maxc1):

HowToUse.bravo

> How to access IFS, and administrative information on the Parc IFS.

Operation.bravo

> Instructions for operating the IFS software.

IFSScavOp.bravo

> Instructions for operating the IFS Scavenger program.

IFSFileStructure.bravo

> IFS file system specification (how files are organized on the disk).

IFSDirOps.bravo

> Description of the BCPL interface to the IFS directory modules implementing the file system.

IFSScavDesign.bravo

> Design notes for the IFS Scavenger program.

Most of these documents are also available in Press and Ears format.

Future Plans

The interim nature of the IFS should be emphasized. The IFS is not itself an object of research, though it may be used to support other research efforts such as the Distributed Message System. We hope that Juniper will eventually reach the point at which it can

replace IFS as our principal shared file system.

That disclaimer having been made, here are the additions we are considering:

> Archiving to magnetic tape via the Alto Magtape system. (However, there is some possibility that archiving to disk packs will prove to be economically feasible.)
>
> Page-level, transaction-oriented operations in the style of WFS (Woodstock File Server).
>
> A mail server for the Distributed Message System.
>
> Low-level access to many of the IFS primitives, permitting improved user interfaces to be constructed (e.g., a DDS-like program for managing files on IFS).

A separate memo describing our immediate plans is available as <IFS>StatusAndPlans.bravo.

Inter-Office Memorandum

To	IFS Users	Date	July 17, 1979
From	Ed Taft and David Boggs	Location	Palo Alto
Subject	How to Use IFS (version 1.21)	Organization	PARC/CSL

XEROX

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

Interim File Systems are now in operation at a number of sites. This memo documents the IFS facilities available to most users.

The names of some of the IFSs presently operating are as follows:

Name	*Organization*
Ivy	Parc (Palo Alto)
Iris	SDD (Palo Alto)
Isis, Sun	SDD (El Segundo)
Ibis	ASD (Palo Alto)
Oly	ASD (El Segundo)
XEOS	EOS (Pasadena)
ADL	ADL (El Segundo)
Erie	WRC (Webster)
Eagle	Corporate headquarters (Stamford)

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

This edition describes IFS version 1.21. Important changes since the previous version include:

addition of a mail server and a mail forwarding capability;

direct transmission of Press files to printing servers;

addition of a Directory command in the Chat server Executive;

ability for individual users to 'own' protection groups and change their membership without intervention by an IFS administrator;

'Print Directory-Parameters' command renamed to 'Show Directory-Parameters'.

Administration

This section applies specifically to Ivy, the Parc IFS. IFS systems at other sites are administered independently.

We we will provide Ivy accounts on request to any member of the Palo Alto community (Parc, SDD, or ASD) who presently has a Maxc account. You must have an Ivy account in order to store files of

your own or to access files stored on Ivy by other users.

To obtain an Ivy account, simply send a message to Ed Fiala. You should include in this message the password you would like installed (the password can but need not be the same as your Maxc password, and you are free to change it yourself at any time).

Personal accounts for Parc users will be assigned a disk limit of 1000 pages, and this limit will be enforced strictly at all times. (One IFS page is approximately equivalent to one Maxc page or four Alto pages.) Our intention is not to oversubscribe the actual storage capacity of the file system at any time, thereby avoiding the extreme storage crunches that have been encountered on Maxc.

Allocations for project (file-only) accounts should be negotiated with Ed Fiala. Please limit requests for storage to immediate rather than long-term needs. If a files-only directory is to be maintained by several users, you may request that a user group be assigned; you should designate who is to be permitted to change the membership of the group.

In general, non-Parc users of Ivy will be assigned a disk limit of zero, and no non-Parc files-only directories will be created. Such users are expected to use their own organizations' IFS systems for file storage.

Communication of files among users of different IFSs is a problem, since most people will have accounts on only one server. Our interim solution to this problem is as follows:

1. Commonly-accessed files (e.g., the contents of the <Alto> directory) will be maintained on each IFS. It will be the responsibility of the administrator of each system to keep such files up-to-date, and to move large volumes of data only during off-peak hours.

2. Users outside Palo Alto will be able to obtain accounts on Ivy (with no storage rights) only under special circumstances. We prefer to limit non-Palo Alto usage because access to an IFS via the slow (9600 baud or less) lines can seriously degrade its availability to local users.

Better long-term solutions are presently under development.

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. Chat may or may not do this for you, depending on the version of Chat you are using.

@ 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.

@@ Host *host-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 '#').

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 'Host' 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.

There are presently no facilities for retracting printing requests or interrogating their status. Also 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, 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 administration.

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.

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, 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

Note: at the time this memo was written, the facilities described in this section had not yet been put into operation at all sites. You should await an announcement from your local support organization before attempting to use these facilities.

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), and HENR (Henrietta). 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 Maxcl); 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'.

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. We are presently considering how much additional effort to invest in IFS development. This topic is covered in some detail in a separate memo, available as [Maxc1]<IFS>StatusAndPlans.bravo and .press.

A major concern is that of performance of the file system. There is insufficient capacity (particularly main memory) in the IFS Alto to support more than a small number of simultaneous users. While we expect eventually to effect some improvements by better implementation and fine-tuning, there is little prospect for really major performance gains.

We are presently imposing a limit of five concurrent connections (FTP and Chat users combined), at which point 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. (The limited number of concurrent servers is one of the reason we wish to discourage large-scale file transfer activity on the Parc IFS by users outside Palo Alto, since the low speed of the communication links causes such transfers to take a long time to complete, thereby tying up servers.)

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 theTrident disk hardware, microcode, and software to make it work reliably. And Steve Butterfield implemented all the Mail facilities and made several other improvements.

Inter-Office Memorandum

To	IFS Project	Date	July 17, 1979
From	Ed Taft	Location	Palo Alto
Subject	IFS Operation (version 1.21)	Organization	PARC/CSL

XEROX

Filed on: <IFS>Operation.bravo, .press

This memo describes operating procedures for the Interim File System software. It is assumed that the reader is familiar with standard Alto software in general and with IFS from the user's point of view (<IFS>HowToUse.bravo).

A short summary of revisions to this document may be found at the end.

The current release of IFS should be run under OS version 16. When OS 17 is released, IFS will run correctly under it also.

1. Hardware requirements and organization

IFS requires a standard Alto with a Trident disk controller and one to eight Trident T-80 or T-300 disk drives in any combination. A T-80 disk pack holds 36,675 pages of 1024 words each, while a T-300 holds 139,365 pages. Due to a software limitation, only 130,986 pages of a T-300 are accessible.

The software deals with a *primary* file system consisting of one or more disk packs. A multiple-pack IFS behaves logically as a single file system, and all packs must be present and on-line in order to access any files. Files created by IFS are not accessible to other Trident-based programs (e.g., TFU, FTP) or vice versa.

Additionally, the software can deal with one or more *secondary* file systems that may be mounted and dismounted while IFS is running. An example of a secondary file system is a disk pack used for on-line, incremental backup. We envision possible other uses for secondary file systems in the future (e.g., large dictionaries for spelling-correction services). This capability is not yet completely implemented.

The number of disk drives needed to support a given size file system depends both on how backup is to be accomplished and what degree of redundancy is desired in case of disk drive failure. Backup procedures are discussed in a later section.

2. Obtaining IFS software

The software is presently distributed from the <IFS> directory on Maxc1. It consists of the following files:

IFS.Run, the program itself.

IFS.Syms, for debugging by means of Swat.

IFS.Errors, mapping error numbers to strings.

IFSScavenger.Run, the IFS Scavenger.

IFSScavenger.Syms, symbols for the IFS Scavenger.

The file IFS.Ov, included with IFS releases before version 1.18, is no longer required. You should delete any copies that you have.

These should be installed on a fairly clean Alto disk, along with standard Alto subsystems such as FTP. Additionally, the following programs may be useful (obtained from the <Alto> directory):

TFU.Run, for formatting and testing Trident disk packs.

Triex.Run, for checking out the Trident disk hardware.

CopyDisk.Run, for copying disks.

All announcements of new IFS releases are made to the distribution list <Secretary>IFSAdministrators.dl. Such announcements include the version number of the system (e.g., 1.03); this is the first number printed out in the IFS herald when you connect using FTP or Chat. Be sure to obtain all three of the IFS.* files listed above.

It is important that you use the version of IFSScavenger appropriate to the version of IFS you are running. The IFSScavenger has intimate knowledge of the file system format, which sometimes changes in minor ways from one release to the next.

3. Testing disk hardware and disk packs

A file system should be built on a set of disk packs that have been thoroughly tested using the 'Certify' command in the latest (November 9, 1977 or later) release of TFU. This procedure is essential for reliable operation. The procedure for each pack is as follows.

Mount the pack on some drive, *d*. Then issue the command:

TFU Drive *d* | Certify

TFU will display the message 'Confirm wiping the pack on drive *d*', to which you should respond 'OK'.

TFU will now initialize the headers on the pack, then execute 10 passes of writing and reading random data on the entire surface of the pack. Any bad spots that it finds are recorded permanently on the pack in a place that IFS and other Trident software can find. IFS will simply avoid using pages known to contain bad spots. The running time for this test is 27 minutes for a T-80 pack and 91 minutes for a T-300 pack. If possible, it is recommended that the test be run for more than 10 passes. This is accomplished by the command 'TFU Drive *d* | Certify *n*', where *n* is the desired number of passes. The running time is approximately 3 minutes per pass on a T-80 and 9 minutes per pass on a T-300.

Before IFS is run in a new installation, it is a good idea to exercise the hardware thoroughly. The TFU program has an 'Exercise' command that exercises the hardware thoroughly in a manner similar to IFS. The basic procedure is as follows.

1. Mount scratch packs on all Trident disk drives (including backup and spare drives).

2. If the packs haven't already been certified, use 'TFU Certify' to initialize the headers on all packs, as described above.

3. For each drive, issue the command 'TFU Drive *d* | Erase'. TFU will display the message 'Confirm wiping the pack on drive *d*', to which you should respond 'OK'.

3. Issue the command 'TFU Exercise *n*'.

This begins a lengthy exercise procedure that creates, reads, writes, and copies files on all packs. *n* is the number of passes to execute. Each pass takes about 20 minutes per T-80 and 60 minutes per T-300 being exercised. TFU writes a log file Tfu.ExerciseLog on the Diablo disk (which should be quite empty), at the end of which should appear the message 'There were 0 errors' when TFU is done.

For further information on the operation of TFU, consult the Trident software documentation in the Alto Subsystems Manual or in <AltoDocs>Tfs.tty.

4. Initializing a file system

IFS is invoked by the command

IFS/switches

where the switches control the operation of the system in various ways. Switches defined at present are:

/C Create a new file system (see below)

/D Debug mode (various non-fatal errors call Swat rather than just continuing on).

/A Allocator debug (every call to the storage allocator causes a very thorough consistency check to be invoked. This slows down operation of the system considerably.)

/B Enable calls to Block within the disk driver, thereby permitting more overlapping of computation with disk transfers. The default state of this switch is true (it may be turned off by '/B').

/V Verify the structure of the directory B-Tree when the system is started (see section 5). The default state of this switch is true.

/X Use extended memory, if present, for caching code overlays. This considerably reduces disk thrashing under heavy load. The default state of this switch is true.

/S Create a buffer (spyBuffer) for use by the Swat Spy facility, for performance measurements.

/M Enables the 'miscellaneous' servers (name, time, and boot). They are ordinarily enabled and disabled by means of the privileged Change System-parameters command (described later), but /M or /M will override the effects of that command for one invocation of IFS. (See section 9.)

When IFS is started with the /C switch, it enters a dialogue in which you must supply various file system parameters. The dialogue takes the following form.

Do you really want to create a file system?

Answer 'y'.

Number of disk units: STARTS FROM "0"

Type in the number of disk units to be included in the file system, terminated by Return.

Logical unit 0 = Disk drive:

Type the physical unit number of the drive that is to be logical unit 0 in the file system. This question is repeated for each logical unit in the file system.

File system name:

Enter the name of the file system, followed by Return. This name is displayed as the first part of the herald generated by the file server and Executive, and should be something like 'Parc IFS'.

Directory size (pages):

This determines the number of disk pages to preallocate for the directory. IFS will suggest a number (based on the number of disk units) which you may confirm by typing Return; or you may type some other number followed by Return. To be conservative, you should specify 1000 times the number of disk packs you ever expect to include in the primary file system. (See section 10.)

Ok? [Confirm]

Answer 'y' if you want to go ahead, or 'n' if you made a mistake and wish to repeat the dialogue.

IFS now initializes the file system, an operation that takes about 2 minutes per T-80 and 7 minutes per T-300 in the system. When the screen turns black and the cursor changes from an hourglass to 'IFS', initialization is complete.

The file system initially has only three directories defined: System, Default-User, and Mail. The password for System is IFS. You should connect to the IFS using Chat, login as System (password IFS), create some other users (by means of the Create command, described below), and change System's password for the sake of security.

Before putting the system into service, there are various system parameters you must set; these are described in later sections. They include:

clock correction and server limit (section 7);

switches to enable or disable the mail system, Press printing, and the boot, name, and time servers (sections 7 and 9);

backup system parameters (section 11).

5. Normal operation

The system is normally started simply by invoking IFS with no switches. All file system packs must be mounted and on-line. It is unimportant which packs are mounted on which drives, since the software reads each pack to discover what the system configuration is. (However, there must be no other primary IFS packs on-line. After copying an IFS pack with CopyDisk, you should be careful to remove the copy from the system.)

If IFS fails to start up properly, it will call Swat with an appropriate error message. This will occur, for example, if all necessary disk drives are not on-line. While IFS is starting up, the cursor contains an hourglass; this changes to 'IFS' when startup is complete and the system is in operation.

When IFS is started, it verifies the consistency of the directory B-Tree (unless inhibited by '/V'). Inconsistencies can result from crashes at inopportune moments when the B-Tree is in the process of being modified, so the startup-time check is valuable in determining whether it is appropriate to run the IFS Scavenger. The time required for the check is proportional to the number of files in the system; it has been observed to take 2 minutes for 20,000 files.

If an inconsistency is detected, IFS will call Swat with an appropriate message. The message 'Record count disagrees' is relatively benign and it is reasonably safe to proceed from this (with control-P). If you do so, it is likely that one or more files in the file system will become inaccessible and will remain so until the next time the IFS Scavenger is run. Other possible errors include 'Records out of order' and 'Malformed B-Tree record'; these are more serious and proceeding is *not* recommended.

Immediately after IFS starts, it will perform some other initialization operations requiring a lot of disk activity, including obtaining and installing the network directory (name server data base) and boot files. IFS can service user requests during this time (5 minutes or more), but performance will be noticeably poorer than normal.

While IFS is running, the entire screen is black except for the cursor. The position of the cursor is an indication of disk activity. The horizontal position indicates the disk unit most recently accessed (unit zero at the extreme left, unit seven at the right), and the vertical position is the cylinder at which the heads are currently positioned (zero at the top, 814 at the bottom). The cursor blinks each time a page is transferred to or from a user file by the file server.

You can tell whether IFS is running normally by pressing the space bar. If the Alto screen flashes, everything is in order; if not, the system has failed in some way.

There are two ways to stop the system. The normal (and cleaner) way is to connect to IFS using Chat, log in, Enable, and issue the Halt command. The system will then refuse to admit further users, will wait for all present users (including you) to log out or disconnect, and will return control to the Alto Executive.

The system may also be stopped by typing Shift-Swat on the IFS Alto keyboard (the Swat key must be pressed firmly). This aborts all active connections and returns control to the Executive immediately; however, it may leave partially-written files lying around so it is not recommended for normal use.

6. User account management

The IFS Executive (accessed via Chat) has several privileged commands that are available only to users with the 'wheel' capability, and only after enabling this capability with the Enable command. When you are so enabled, the Executive's prompt is '!' rather than '@'. While you are enabled, IFS will not log you out automatically after three minutes of inactivity as it does normally.

The following privileged commands are defined.

! Create (directory) *directory-name* [new] (password) *password*
!! *sub-commands*

Creates a directory with the supplied name and password. Capitalization of the name should be precisely as the user wants to see it. Capitalization of the password is unimportant.

The sub-commands are used to change various parameters from their default values. These include:

!! Files-only (owner) *user-name*

Declares the directory to be a files-only (i.e., non-login) directory. The user-name is the person responsible for this directory (used for administrative purposes only).

!! Disk-limit *number*

Specifies the maximum number of disk pages that may be used in this directory.

!! Wheel

Declares the user to have the 'wheel' capability, which permits issuing privileged commands and bypasses all access checking and disk limits.

!! Mail

Creates a mailbox, thereby enabling IFS to receive Laurel mail for this user. (More information about mail is presented in section 8.)

If *directory-name* already exists, you are not asked for a password but rather are sent directly into subcommand mode. This permits modifying parameters for an existing directory. In this context, the following additional subcommands may be of interest:

!! Password *password*

!! Not Files-only

!! Not Wheel

!! Not Mail

The Create command is terminated by typing two Returns in response to the '!!' subcommand prompt. You may cancel the entire command by typing control-C.

The default values of all parameters for new directories are copied from a ficticious directory called Default-User. When a file system is created, the default values are Not Files-only, Not Wheel, Not Mail, and Disk-limit 1000. To change the defaults, use the Create command to modify the parameters for the directory Default-User. 'Not Mail' is the default for files-only directories, even if Default-User specifies 'Mail'.

The Create command does not provide facilities for setting default file protections or directory protections; these are accessible only via the Change Directory-Parameters command. Sorry about that.

! Destroy (directory) *directory-name* [Confirm]

Destroys the specified directory. This operation includes deleting all the files contained within it, and destroying the associated mailbox if there is one.

! Change Directory-Parameters (of directory) *directory-name*
! Show Directory-Parameters (of directory) *directory-name*

These commands work as described in the 'How to Use' document with the addition that while you are enabled, you may access any directory, not just your own. Also, while you are enabled, Change Directory-Parameters has the following additional sub-commands:

!! Group Membership (in groups) *groups*
!! Group Ownership (of groups) *groups*
!! No Group Membership (in groups) *groups*
!! No Group Ownership (of groups) *groups*

These permit you to establish a user's membership in or ownership of user groups. (The Group Membership sub-command duplicates the function of the top-level Change Group-Membership command.)

7. Other privileged commands

! Disable

Leaves enabled mode (the Executive's prompt reverts to '@').

! Halt

Stops IFS and returns control to the Alto Executive as soon as all present users (including you) log out or disconnect.

! Change System-Parameters
!! *sub-commands*

Permits you to issue sub-commands to change one or more system operating parameters. Each sub-command takes effect immediately. Sub-command mode is terminated when you type CR in response to the '!!' prompt.

The following sub-commands have permanent effects that survive restarts of IFS.

!! Clock-Correction *correction*

Sets the software clock correction, which is specified as a sign (+ or -) followed by a decimal number. This causes the Alto clock to run faster (+) or slower (-) than its nominal rate by that number of seconds per day. Alto clocks are quite stable but not particularly accurate, and software correction is desirable in a server that runs continuously for long periods of time.

The amount by which the clock should be corrected may be determined by comparison with an accurate reference over a period of several days (using the IFS Executive's 'DayTime' command), or by use of an accurate frequency counter to measure the Alto system clock (slot 5 pin 63 on an Alto-I, slot 13 pin 63 on an Alto-II) and computing the correction by the formula

$$c = 86400 * (1 - f / 5880000)$$

where f is the frequency in Hz.

!! Server-Limit *n*

Limits the number of simultaneous server processes (FTP, Chat, and Mail combined) to *n*. At present, *n* may be between 1 and 6, and the default is 5. Depending on use patterns, for some IFSs 5 simultaneous servers result in unacceptably poor performance or occasional deadlocks (see section 13), and it may be desirable to reduce the limit to 4.

!! Enable Press-printing
!! Enable Boot-server
!! Enable Name-server
!! Enable Time-server

!! Disable Press-printing
!! Disable Boot-server
!! Disable Name-server
!! Disable Time-server

Enable and disable the Press printing facility and the various miscellaneous servers (see section 9). When a file system is created, all the servers are disabled.

!! Enable Mail System
!! Enable Mail Forwarding

!! Disable Mail System
!! Disable Mail Forwarding

Enable and disable the mail system (see section 8). The first command turns on and off the mail system as a whole; the second command enables or disables forwarding of mail to other mail servers.

!! Dead-letter (recipient name) *name*

Specifies the name of the mailbox to which notification of mail system problems (e.g., undeliverable messages with no return address) should be directed. *Name* may be the name of a mailbox on this IFS or (if forwarding is enabled) the fully-qualified name (*'user . registry'*) of a recipient on some other server.

The following sub-commands have one-time-only effects.

!! Disable Logins

Disallows further user access to the system. This is useful during debugging and while reloading the file system from backup.

!! Enable Logins

Cancels the effect of Disable Logins.

!! Reset-Time

Causes IFS to reset its clock from a time server on the directly-connected Ethernet. This operation is performed once automatically, immediately after IFS restarts.

8. Mail server

IFS contains a mail server that is compatible with the Laurel message system. IFS can keep mailboxes for users of that system and can also forward mail to mail servers in other IFSs and in Maxc.

To enable a user to receive mail, issue the Mail subcommand of the Create command, as described previously. (Of course, if you turn on the Mail capability for Default-User, then all new user accounts you create subsequently will have Mail capability automatically.)

When mail is received from a Laurel user, it is queued briefly in files named '<Mail>New>Mail!*'. A process called Mailer then wakes up and distributes the messages to individual in-boxes, which are files named '<Mail>Box>*user-name*!1'. When a user gets the mail from his in-box, the in-box file is reset to empty (but is not deleted).

The Mailer process also forwards mail to other mail server hosts. Specifically, messages addressed to *user.host*, where *host* is the name of some other mail server host, will get forwarded to that host by the Mailer process. While being forwarded, such messages are queued in files named '<Mail>Fwd>*host*'.

Important: Existing file systems created by IFSs earlier than version 1.18 do not have a <Mail> directory. To enable the mail server to operate, it is necessary to set up such a directory with the proper attributes:

```
! Create (directory) Mail [new] (password) random-password
!! Files-only (owner) System
!! Disk-limit number [If this is ever exceeded, the mail server will stop working]
!! [Confirm] yes
! Change Directory-parameters (of directory) Mail
!! No Read (access permitted to) World
```

!!

The IFS version 1.21 mail system is incompatible with that of earlier IFS releases. The names of working files in the <Mail> directory have intentionally been made different from the old ones in order to avoid difficulties. Consequently, if you ran a mail system under an earlier release of IFS, you should request that users clean out their mailboxes before you convert to IFS 1.21. Files named '<Mail>*user-name*' are those used by the old IFS and will never be referenced by IFS 1.21. You should delete them as soon as you are certain they are empty.

Also, you will have to re-establish the mailboxes for each of your existing mail users, using the 'Mail' subcommand of the 'Create' command.

When a file system is first created or reloaded from backup, the mail system is disabled. You should set the enable switches and establish the dead-letter recipient name using the 'Change System-Parameters' command (section 7).

9. Miscellaneous servers

IFS contains name, time, and boot servers that provide essential services to other hosts (principally Altos at present) on the directly-connected Ethernet.

These functions duplicate those provided by gateway systems, so in a network with at least one gateway, it is not necessary for IFS to provide these services. But in a network that includes an IFS and no gateways, it is necessary for the IFS to provide the services. Even in networks that do have one or more gateways, running the IFS miscellaneous servers may be advantageous in that the availability of the services is improved. (Also, it should be noted that the IFS boot server is noticeably faster than the boot servers of existing gateway systems.)

Since running the miscellaneous servers may slightly degrade the performance of an IFS in its principal functions, means are provided to turn them off (the Change System-Parameters command, described in section 7). When a file system is first created, all the miscellaneous servers are disabled.

IFS participates in the protocols for automatic distribution and maintenance of the date and time, the network directory, and the common boot files. When IFS is started up for the first time, and thereafter whenever any changes are distributed, IFS obtains all necessary files from neighboring servers (gateways or other IFSs). The name server data base is maintained even if the IFS name server is disabled, because IFS requires it for its own internal purposes (principally mail forwarding).

The name server data base is kept as file '<System>Pup-network.directory'; a new version is created and older versions deleted whenever a new file is distributed. The boot files are kept in files '<System>Boot>*number-name*.boot', where *name* is the name of the boot file and *number* is its boot file number in octal (for example, '<System>Boot>4-CopyDisk.boot'). Standard boot files have centrally-assigned boot file numbers less than 100000 octal, and are distributed automatically. IFS will obtain its own copy of all standard boot files maintained by any other boot server on the same Ethernet. Non-standard boot files have boot file numbers greater than or equal to 100000 octal and are not distributed automatically.

The best way to install a new version of one of these files is to use the facilities provided by the GateControl and MakeDirectory programs. These programs use the same update protocols that the boot servers use for communication among themselves. If you store files by other means (e.g., FTP), you must then restart IFS to cause those files to be installed and recognized by the servers.

You should not enable the time server unless you have first calibrated and corrected the Alto clock, using the procedure described in section 7.

10. Adding packs to the file system

The capacity of an existing file system may be increased by adding more packs to it. This may be accomplished by the following procedure.

Initialize and test a pack using 'TFU Certify' in the normal fashion (section 3). Then, with IFS running, mount that pack on any free drive and issue the command:

! Extend (file system) Primary (by adding drive) *d* [Confirm]

'Primary' is the name of the file system you are extending, and *d* is the drive on which the new pack is mounted. IFS now initializes this pack, an operation that takes about 2 minutes for a T-80 and 7 minutes for a T-300. When it completes, the new pack has become part of the file system.

Note that there is no corresponding procedure for removing a pack from a file system. To decrease the number of packs in a file system, it is necessary to dump it by means of the backup system, initialize a new file system, and reload all the files from backup. This procedure is also required to move the contents of a file system from T-80 to T-300 packs.

Note also that adding packs to a file system does not increase the amount of directory space available. The size of the directory is determined when you first create the file system; there is no straightforward way (short of dumping and reloading) to extend it. (More precisely, while the software will attempt to extend the directory automatically if it overflows, this will significantly degrade subsequent performance, and too many such extensions will eventually cause the system to fail entirely.) Therefore, it is important that you allocate a directory large enough for all expected future needs. Experience has shown that 1000 directory pages are required for every 25,000 files in the file system, but this is highly dependent on a number of parameters including average file name length.

11. Backup

There are three facilities available for assuring reliability of file storage and for recovering from various sorts of disasters.

The first facility is the IFSScavenger program. It is analogous to the standard Alto Scavenger subsystem. It reads every page in the file system, makes sure that every file is well-formed, and checks for consistency between files and directories. For safest operation, it should be run after every crash of the IFS program. However, since it takes a long time to run, in practice it should only be run when major file system troubles are suspected (in particular, when IFS falls into Swat complaining about disk or directory errors). The IFSScavenger is described in a separate memo, available as <IFS>IFSScavOp.Bravo (or .Press) on Maxc1.

The second facility is an on-line incremental backup system that is part of the IFS program itself. It operates by copying files incrementally to a backup file system mounted on an extra drive. The file system is available to users while the backup operation is taking place (though backup should be scheduled during periods of light activity to avoid serious performance degradations). Use of the incremental backup system requires that there be an additional disk drive connected to the Alto, over and above the drives needed for the primary file system itself. The backup system is described in the next section.

The third facility is the CopyDisk program. To back up the file system, one must take IFS down and copy each of the file system packs onto backup packs. On a machine with multiple disk drives, one may copy from one drive to another, an operation that takes about 4 minutes per T-80 and 15 minutes per T-300 if the check pass is turned off. One may also copy disks over the Ethernet to another Alto-Trident system, but this takes about five times as long.

At Parc we use the Scavenger and the Backup system; we no longer use CopyDisk for backing up IFS. Regular operation of either the Backup system or CopyDisk is *essential* for reliable file storage.

We have observed several instances of Trident disk drive failures that result in widespread destruction of data. It is not possible to recover from such failures using only the IFS Scavenger: the Scavenger repairs only the *structure* of a file system, not its contents.

11.1. Backup system operation

The backup system works in the following way. Periodically (e.g., every 24 hours), a process in IFS starts up, checks to make sure a backup file system is mounted, and sweeps through the primary file system. Whenever it encounters a file that either has never been backed up before or was last backed up more than *n* days ago (a reasonable *n* is 30), it copies the file to the backup file system and marks the file as having been backed up now. Human intervention is required only to change backup packs when they become full.

The result of this is that all files are backed up once within 24 hours of their creation, and thereafter every *n* days. Hence every file that presently exists in the primary file system is also present in a backup file system written within the past *n* days. This makes it possible to re-use backup file system packs on an *n*-day cycle.

Operation of the backup system has been made relatively automatic so as to permit it to run unattended during the early morning hours when the number of users is likely to be small. This is important because system performance is degraded seriously while the backup system is running.

11.2. Initializing backup packs

To operate the backup system, you need a disk drive and some number of packs dedicated to this purpose. The number of packs required depends on the size of your primary file system, the file turnover rate, and the backup cycle period *n*. The packs should have their headers and labels initialized using 'IFU Certify' in the normal fashion. Then they must each be initialized for the backup system as follows.

With IFS running, mount a backup pack on the extra drive. Connect to IFS from some other Alto using Chat, log in, enable, issue the Initialize command, and go through this dialogue:

! Initialize (file system type)

Answer 'Backup'.

Do you really want to create a file system?

Answer 'y'.

Number of disk units:

Answer '1'.

Logical unit 0 = Disk drive:

Type the physical unit number of the drive on which the backup pack is mounted.

File system ID:

Type some short name that may be used to uniquely identify the pack, e.g., 'Backup1', 'Backup2', etc. No spaces are permitted in this identifier. It should be relatively short, since you will have to type it every time you mount the pack. (You should mark this name on the pack itself, also.)

File system name:

Type some longer identifying information, e.g., 'Parc IFS Backup 1', or 'Serial number xxxx', or something.

Directory size (pages):

Type Return. (The default of 1000 pages is plenty.)

Ok? [Confirm]

Answer 'y" if you want to go ahead, or 'n' if you made a mistake and wish to repeat the dialogue.

IFS now initializes the backup file system, an operation that takes about 2 minutes for a T-80 and 7 minutes for a T-300. The message 'Done' is displayed when it is finished.

11.3. Setting backup parameters

The next step is to set the backup parameters, an operation that generally need be done only once. Issue the Backup command to enter the backup system command processor (whose prompt is '*'), then the Change command. It will lead you through the following dialogue:

* Change (backup parameters)
Start next backup at:

Enter the date and time at which the next backup run is to be started, in the form '7-Oct-77 02:00'.

Stop next backup at:

Enter the date and time at which the next backup run is to stop if it has not yet completed, e.g., '7-Oct-77 05:00'.

Interval between backup runs (hours):

Type '24'.

Full file system dump period (days):

Enter the number of days between successive backups of existing files (the parameter *n* above). A good value is 30.

The backup system command processor is exited by means of the Quit command in response to the '*' prompt.

11.4. Normal operation

The following commands are used during normal operation. All of them require that you first Enable and enter the backup system command processor by means of the Backup command.

* Status

Prints a message describing the state of the backup system. It will appear something like:

```
Backup system is enabled and waiting.
Backup scheduled between 7-Oct-77 02:00 and 7-Oct-77 05:00
File system Backup1 is available to backup system.
73589 free pages.
```

'Enabled' means that at the appropriate time the backup system will start up automatically; the opposite is 'disabled'. The backup system becomes enabled when you mount a backup pack (see Mount command, below), and disabled when the backup system can no longer run due to some condition such as the backup file system being full.

'Waiting' means that the backup system is not presently running; the opposite is 'running'. When it is running (or has been interrupted in the middle of a backup run for whatever reason), it will display an additional message of the form:

Presently working on file *filename*

as an indication of progress (files are backed up in alphabetical order).

The last lines display the status of the current backup file system (assuming one has been mounted. If several backup file systems have been mounted, they will all be listed.) The possible states are 'available', 'presently in use', and 'no longer usable'. In the last case, the reason for the non-usability is also stated, e.g., 'Backup file system is full'.

* Enable (backup system)
* Disable (backup system)

Enables or disables running of the backup system. If Disable is issued while the backup system is actually running, it will stop immediately (within a few seconds). These commands are not ordinarily needed, because an Enable is automatically executed by Mount (see below) and a Disable is executed when the backup system finds that there are no longer any usable backup file systems. The backup system also stops automatically if IFS is halted by the Halt command, but it is not disabled and will resume running when IFS is restarted.

* Mount (backup file system) *name*

Makes a backup pack known to the system. *name* is the file system ID of the backup file system (e.g., 'Backup1'). The pack must be on-line.

If the file system is successfully mounted, a message appears in the form:

Backup1 (Parc IFS Backup 1),
initialized on 6-Oct-77 19:32, 273 free pages.
Is this the correct file system? [Confirm]

If this is the file system you intend to use, you should answer 'y'. Then:

Do you want to overwrite (re-initialize) this file system? [Confirm]

Normally you will be mounting a backup file system that has either never been used before or was last used more than *n* (e.g., 30) days ago. In this case you should answer 'y'. This will cause the backup file system to be erased (destroying all files stored in it) at the beginning of the next backup run.

If, however, you are re-mounting a partially-filled backup file system that was removed for some reason, you should answer 'n'. The backup system will then not erase the backup pack but rather will simply copy additional files to it.

* Dismount (backup file system) *name*

Makes a previously mounted backup file system unavailable to IFS. This command may be issued only while the backup system is disabled (use the Disable command if necessary).

The normal operating procedure is very simple. Every day, issue the Enable and Backup commands to enter the backup system command processor, then issue the Status command. The status will indicate one of the following conditions:

1. 'Enabled and waiting', with one or more file systems 'available to backup system'. In this case you need not do anything.

2. 'Disabled and waiting', with one file system 'no longer available to backup system' because 'Backup file system is full'. In this case, you should remove the backup pack, install another one (making sure it was last used more than *n* days ago), and declare it to IFS by means of the Mount command (above).

3. 'Disabled and waiting', with some other condition (e.g., 'Can't find logical unit 0'). You should correct the condition (most likely the required pack wasn't mounted at the time the backup system last started to run), then issue the Mount command as above.

When done, issue the Quit command to exit the backup system command processor. It is a good idea to keep a record of the dates on which each backup pack was mounted and dismounted so that you know when a pack is available for re-use.

11.5. Restoring individual files from backup

Individual files may be restored from backup in the following manner. It is not a good idea to do this while the backup system is running.

Install the desired backup pack on any free drive. Issue the Enable and Backup commands to enter the backup command processor. Then go through the following dialogue:

> * Restore (from file system) *name*
> *name* (*long-name*) mounted
> Restore: *file-designator*

The *name* is the File system ID of the backup pack (e.g., 'Backup1'). In response to 'Restore:', type the name of a file to be restored. '*'s are permitted, and the default version is '!*'. The name of each file is typed out as it is restored.

When all files matching *file-designator* have been restored, IFS will again prompt you with 'Restore:'. You may either restore more files (from the same backup file system) or type Return to indicate that you are finished.

Files are restored from the backup system with precisely the attributes (version number, reference dates, etc.) they had when backed up. If a file already exists in the primary file system, IFS will refuse to overwrite it unless the version in the backup file system is *newer.*

11.6. Reloading the entire file system from backup

If the primary file system is clobbered in a way that the Scavenger can't repair, the following procedure may be used to recreate it from backup. If performed correctly, this procedure will restore the primary file system to its exact state at the time of the most recent backup run.

First, re-initialize the primary file system as described earlier (section 4). Then connect to IFS from another Alto using Chat, login as System (password IFS), and issue the Enable command. It is advisable at this point to disable logins with the Disable Logins subcommand of the Change System-parameters command so as to prevent users from accessing the file system while you are reloading it.

Mount (on any free drive) the *most recent* backup pack, i.e., the one most recently written on by the backup system (this is very important). Then:

```
* Reload (file system)
Note: mount the LAST backup file system first.
Mount file system: name
```

The *name* is the ID of the backup file system you have mounted. IFS will now proceed to restore files from the backup file system to the primary file system. When it is done, it will again ask you to 'Mount file system:', at which point you should mount the next most recent backup pack. Repeat this procedure until you have mounted all packs written within the past *n* days.

IFS will list out the files as they are restored. (To disable the pause at the end of each page, type ahead one space.) You will notice that not all files are restored. In particular:

Files that were backed up at some time but no longer existed at the time of the last backup are not restored. (The listing will say such a file is 'deleted'.)

Files already restored from a more recent backup are not restored from an earlier one. (The listing will say 'already exists'.)

Reloading the file system causes all backup and system parameters to be reset. You must set them up again manually. See the summary at the end of section 4.

It is *essential* that the last backup pack be reloaded first. Failure to heed this instruction will cause some files not to be reloaded that should have been, and vice versa. If the reload is interrupted for any reason and must be restarted, you must again start by reloading the last backup pack (even though all files from that pack may have been reloaded already). This is because the decision whether or not to reload each file is made on the basis of the last state of the file system as recorded on the most recent backup pack.

12. Accounting

Accountant.run is a program which collects accounting and administrative information from a running IFS. It retrieves copies of all of the Directory Information Files (DIFs) from an IFS and produces a text file containing per-directory and system-wide information. To run it type:

```
>Accountant IFS-host-name output-filename
```

You must be a wheel, since the DIFs which Accountant reads are protected. Note that you run this program on some *other* Alto, not on the IFS Alto.

This program is fairly primitive at the moment. There are no present plans to upgrade it.

13. Miscellaneous

13.1. Disk pack identification

If you forget the ID of some Trident pack (e.g. a backup pack), there is no way to 'Mount' it for the backup system. This is why it is a good idea to mark the ID on the pack itself (not on its plastic cover, which is interchangeable with other packs). A good place to mark it is on the plastic ring on the top of the pack. Do *not* affix a paper label: it will fly off and gum up the works (the pack spins at 3600 RPM).

There is, however, a command for finding out vital information about a pack. It is:

```
! What (is the pack on drive) d
```

where *d* is a drive number. If the pack is an IFS pack (primary or backup), this command will print out the vital parameters, including the ID. If the pack is not an IFS pack, it will say so.

13.2. Software performance

The IFS software strains the Alto's capacity significantly, particularly with respect to main memory. In combination with certain deficiencies of the BCPL runtime environment, this leads to rather poor performance (in particular, excessive disk thrashing) when there are more than a few simultaneous users of the system.

Also, there are times when certain data structures and code segments cannot be swapped out. It is possible for the system to deadlock if all of memory is occupied by such immovable objects. The symptom of this is that IFS ceases to respond to requests for service, the Alto screen looks normal (black with 'IFS' in the cursor), and the screen does not flash when you press the space bar. The possibility of deadlocks is the principal reason for imposing a limit of five simultaneous server processes. Most IFSs seldom or never encounter deadlocks with this limit. In those IFSs that do, it is possible to reduce the limit by means of the Server-Limit subcommand of the Change System-Parameters command.

If the IFS Alto has extended memory, the software will use it as a cache for code segments. This improves performance significantly, since it reduces disk thrashing. 128K of memory is sufficient to eliminate practically all code swapping from disk, and no improvement will be gained by having more memory than that. However, use of extended memory in this way does not alleviate the conditions that cause deadlocks, because BCPL can neither execute code nor reference data in more than 64K of memory.

13.3. Interpreting system statistics

The IFS Executive's Statistics command pours out various internal operating statistics, some having to do with hardware and some with software. Most are of interest only to IFS implementors, but all are explained here for completeness.

SmallZone overflows, bigZone overflows, overflow pages

> IFS has a three-level memory storage allocator. SmallZone and bigZone are heap-type allocators for objects of less than 25 words and of 25 to 500 words, respectively. Objects larger than 500 words are allocated by removing one or more 1024-word pages from the VMem (virtual memory manager) pool. If one of the first two zones becomes exhausted, it recovers by borrowing space from the next larger zone.
>
> It is normal to encounter up to 100 or so zone overflows per day, and for there to be a maximum of 2 or 3 VMem pages used to recover from bigZone overflows. More overflows are indicative of the need to change some compile-time parameters. If the 'current' number of overflow pages remains nonzero for any significant length of time, it is indicative of a bug (loss of allocated storage).

Net blocks allocated minus blocks freed

> This is simply the number of memory storage blocks presently allocated. If there is no system activity besides your Chat connection, this should be more-or-less constant. If it increases slowly over time, storage is being lost.

VMem buffers, buffer shortages

> Approximately half of Alto memory is turned over to the VMem package, which manages it as a buffer pool of 1024-word pages and implements a software virtual memory for accessing various objects on the disk, including code overlays, directories, and bit tables. The number of VMem buffers is constant for a given release of IFS.
>
> If the VMem package receives a request that it can't satisfy because all buffers are in use by locked objects (or have been removed to service a zone overflow), it increments the 'buffer shortages' count (*vMemBufferShortages*, accessible from Swat) and then waits, in the hope

that some other process will run to completion and release some buffers. Sometimes this works. On other occasions, all processes in the system get into this state and the system is deadlocked.

VMem reads and writes

This table contains the number of swap reads and writes for each of three main types of objects managed by the VMem package: code overlays, VFile pages (virtually accessed files, principally the IFS directory B-Tree), and DiskDescriptor (disk bit map) pages.

Overlays read from XM and from disk

If the Alto has extended memory, this indicates how many overlay reads have been satisfied by reading from the extended memory cache rather than from the disk.

Disk unit statistics

This table contains operating statistics for each Trident disk unit, and, in the case of T-300 disks, each of the two logical file systems on the unit. All disk statistics are cumulative from the time the file system was created.

File system — File system name and logical unit number within that file system. Logical unit 0 contains the IFS directory and the code swapping region.

Transfers — Number of pages transferred to and from the unit.

Err — The number of errors of all kinds that are not corrected by performing a single retry. Errors not corresponding to the following breakdown are probably data-late errors and can safely be ignored if they are infrequent (no more than 2 or 3 per day).

ECC — The number of data errors detected by the Error Correction Code. These should be extremely infrequent, especially on T-300 drives which are very reliable if properly maintained. (Ivy, the Parc IFS, has not recorded an ECC error in the 9 months since its file system was last reinitialized.) A sudden jump in the rate of ECC errors is grounds for suspicion of a hardware problem.

Fix — The number of ECC errors that have been corrected. The ECC permits correcting error bursts up to 11 bits long.

Rest — The number of times a head-restore operation has been performed. This is done as a last-resort measure when an uncorrectable error persists through 8 retries.

Unrec — The number of errors unrecoverable after 16 retries. This usually causes IFS to fall into Swat and report an unrecoverable disk error. This may be indicative of a hardware problem or of an inconsistency in the structure of the file system. Running the IFSScavenger can tell you which.

BTerr — The number of times the bit table has been found to be incorrect, i.e., to claim that a page is free when it isn't. This in itself is a non-fatal error, but it may be indicative of serious hardware or software problems. On the other hand, it can also be caused by restarting IFS after a crash without first running the IFSScavenger.

Free — The number of free pages on the logical unit. IFS always allocates new files on the emptiest unit, and every file is contained completely within one unit. The software does not recover from running out of disk space (i.e., it falls into Swat), so be careful not to let the amount of free space get too

low.

Directory statistics

These include the number of directory pages assigned and actually in use by the B-Tree package. (The size of the directory file itself, <System>IFS.dir, may be obtained using the List command.) The number of 'runs' (groups of consecutively-allocated pages) should always be 1 unless the directory file has overflowed; a larger number is indicative of fragmentation problems. 'Levels' is the depth of the B-Tree.

Mail statistics

This is a summary of mail server operating statistics, covering the interval since the file system was created or last reloaded.

14. Revision history

Version 1.03; August 4, 1977

Procedures added for running Triex, for blessing your Trident controller, and for halting IFS in a cleaner manner than before.

Version 1.07; September 3, 1977

/V switch added for startup-time directory B-Tree verification.

Version 1.08; October 5, 1977

Backup system released.

Version 1.10; November 1, 1977

Full T-300 support; 'Extend' command for adding packs to an existing file system; Triex and TFU operating procedures changed; IFS Scavenger released.

Version 1.12; November 10, 1977

Performance improvements in both IFS and IFSScavenger; procedures for initializing and testing a disk pack again changed (Triex eliminated from procedure).

Version 1.14; February 21, 1978

File protections implemented; command added to disable logins; backup system bugs fixed.

Version 1.15; March 4, 1978

Converted to new time standard; 'What' command added; automatic SetTime at startup; obscure directory ordering bug fixed.

Version 1.18; November 15, 1978

Mail server added; limited support for extended memory; Change System-Parameters command added, with subcommands to change clock correction, limit the number of simultaneous servers, reset the time, and enable and disable service; Logins command removed; screen flashes if you hit space bar and system is operating normally; Accountant program released; documentation on system performance and interpreting Statistics output; file IFS.Ov is no longer part of the release.

Version 1.21; July 16, 1979

Mail forwarding and Press file printing implemented; miscellaneous servers (name, time, and boot) added; Change System-Parameters subcommands modified; protection groups may now be 'owned' by individual (non-wheel) users; Change Group-Membership and Show Group-Membership commands added to permit users to manipulate group membership; more statistics.

Inter-Office Memorandum

To	IFS Project	Date	July 17, 1979
From	Ed Taft	Location	Palo Alto
Subject	IFS Software Maintenance	Organization	PARC/CSL

XEROX

Filed on: [Maxc1]<IFS>IFSSoftwareMaint.bravo

This is a brief description of how the IFS software is organized and maintained.

Organization

The pieces from which IFS is constructed fall into three major categories:

1. Components specific to IFS; all of these have file names that begin with 'Ifs', and are kept on the master IFS directory (presently [Maxc1]<IFS>).
2. Standard Alto software packages (including pieces of the Alto Operating System), obtained from the <Alto> and <AltoSource> directories. In general, IFS always uses the latest version of these.
3. Modified Alto software packages. In general, these are intended to be released as the standard packages at some later time. Packages that have been so extensively modified for IFS that they are no longer suitable for general release are renamed and moved into category 1.

An 'official' release of IFS consists of the following files, all kept on [Maxc1]<IFS>:

1. IFS.run, IFS.syms, and IFS.errors, the files needed to operate an IFS server.
2. A command file IfsDisk.cm that initializes a blank Alto disk with the minimum set of programs and other files required for IFS development.
3. A dump-format file IfsCm.dm that contains all the command files useful for IFS development.
4. Dump-format files containing all the IFS-specific sources, divided into functional groups. At present, these consist of the following:

IfsDecl.dm	All the Bcpl declaration (.decl) files. These are also contained in the appropriate dump files listed below; they are collected together here because may of them are required throughout the system, not just in one group.
IfsKernel.dm	Basic underlying mechanisms (virtual memory, overlays, storage allocation), plus system initialization.

IfsFileSys.dm	Directory and file access machinery.
IfsRsMgr.dm	Rendezvous socket and server management.
IfsFtp.dm	FTP server.
IfsMail.dm	Mail server and forwarder.
IfsMisc.dm	Miscellaneous servers (name, time, boot, and Press printing).
IfsTelnet.dm	Telnet server and command interpreters.
IfsMc.dm	Microcode.
IfsLeftovers.dm	Everything that doesn't seem to fit into one of the above categories.

5. A dump-format file IfsBrs.dm that contains all the Br files (both IFS code and standard packages) for this release.

The IFS maintenance procedures have been rather carefully organized so that you can do IFS development on a single Diablo disk, with considerable support from one or more file servers. The disk contains all necessary subsystems and all the Br files for loading an IFS, and has enough room (about 500 pages) for a fair number of source files being actively worked on. The idea is that you fetch source files from a file server, modify them, get the modified IFS working again, store the source files back on the file server, and delete them from your Alto disk.

Recreating IFS from sources

This procedure assumes you have access to a file server and directory containing a released version of IFS on a directory called <IFS>.

First, boot an OS from the network and use it to 'erase' a disk. In response to the question 'Do you want a big SysDir' you should answer 'y'.

Next, fetch <IFS>IfsDisk.cm from the file server and execute it. (The command file on Maxcl, naturally, assumes that all standard Alto software should be obtained from Maxcl. If you want to change this you should edit the file first.) The IFS 1.21 release requires a pre-release Alto OS and Bcpl compiler.

Toward the end of this command file, you are asked for the contents of an unknown command file 'FileServerForIfsSoftware', at which point you should type the name of the file server that has all the IFS dump files. IfsDisk.cm now loads IfsCm.dm and IfsDecl.cm.

Before proceeding further, you may wish to modify User.cm, e.g., to change the hardcopy host name.

Now it is time to fetch all the standard and modified software packages (categories 2 and 3). The command file IfsPackages.cm does this. Again, the one on Maxcl assumes you will obtain the standard software from Maxcl. This command file fetches a whole lot of stuff, deletes some of it, and recompiles several packages in non-standard ways. This takes about 30 minutes.

Next, compile all the IFS-specific software by executing the command file CompileIfs.cm. This first asks you what file server the IFS dump files live on. (Your choice must be one of Maxc, Ivy, or Ibis; if you are using some other file server, say 'xxx', you must have previously created a file called 'xxx' whose contents are 'xxx', with no CR at the end.) The command file then loads each dump file in turn, compiles all the source files, and deletes them. This procedure takes about 90 minutes. At the end, you should 'Type *.bt *.er' to check for errors, then 'Delete *.bt *.er'.

Finally, load IFS.run by executing the file LoadIfs.cm. This takes 3 to 4 minutes. There are normally 10 errors (all multiply-defined symbols) which you should ignore. At the end, ListSyms is run to produce IFS.bz, which is a listing of the sizes of the overlays (all except AltoDirs should be 1024 or less words long).

Other command files

Each of the dump files has associated with it a command file that enumerates its contents. That is, IfsKernel.cm contains a list of all the files in IfsKernel.dm. It should not have any CRs in it, even at the end (but of course '↑ CR' is ok). These command files are expanded in many contexts and control loading, dumping, printing, etc., of each group of files.

PrintIfs.cm is a command file that obtains all the source files, group by group, and prints them using Empress.

During active software development, it is inconvenient to have to keep loading and dumping dump-format files; it is more convenient to store all the source files separately, and retrieve, edit, and store them individually.

The command file ExpandIfs.cm loads all the dump files and puts the sources back out as individual files in <IFS>Sources>. You get to specify both source and destination file servers for this operation. (Note that then destination file server must be an IFS because Maxc doesn't have subdirectories. Again, you must previously have created a file whose name and contents are the server name, as described above.)

The command file DumpIfs.cm performs the inverse operation, retrieving all the source files from <IFS>Sources> and dumping them into dump-format files in <IFS>. It also stores Ifs.run, Ifs.syms, Ifs.errors, and IfsBrs.dm from your Alto disk, in preparation for an IFS release.

Command file maintenance

To enable all these mechanisms to work smoothly, it is important to keep the command files up-to-date. Specifically:

> When you add, delete, or rename a source file, you must change LoadIfs.cm, xxx.cm, and Compilexxx.cm, where xxx is the name of the group to which the file belongs.
>
> When you add or remove a package, you must change LoadIfs.cm and IfsPackages.cm.

Inter-Office Memorandum

To	IFS Project	Date	July 29, 1977
From	Ed Taft	Location	PARC/CSL
Subject	IFS File Structure (Edition 2)	File	[Maxcl]<IFS>IFSFileStructure.bravo

XEROX

This document specifies how files are organized in the Interim File System. Included are descriptions of the structure of files and directories, as well as selected algorithms for managing these structures.

Definitions for all file system structures (i.e., those actually stored on the disk) may be found in IFSFiles.Decl, which is an extension to AltoFileSys.D.

Legal IFS Files

A constraint on the design is that the IFS make use of all existing basic file access operations (page-level data transfers, creating and deleting files, disk streams) provided by the Alto OS and TFS package. However, the IFS contains its own special implementations of higher-level functions such as directory management (the existing implementations cannot adequately deal with large numbers of files in a multi-user environment).

All files contained in the IFS, including directories, bit tables, and other "system" files, are Legal IFS Files. A Legal IFS File is a Legal Alto File (see Alto OS manual, BFS description) with additional information recorded in the leader page. This information includes:

Name of file (this is not a hint)
Date and time of last backup
File Protection
Author
etc...

The file leader page is the ultimate authority for the filename and for attributes peculiar to the IFS, such as file protection. This makes it possible to reconstruct the file system if directories are smashed. However, the directory FP and last page FA are still hints, as in the normal Alto file system.

Though the IFS contains several disk units comprising a single logical file system, an IFS file is contained entirely on one disk. This is because much of the code in the TFS and streams packages assumes that a virtual DA will fit in 16 bits; cross-disk addressing would require more than this. However, the virtual DA in a File Pointer (which references the leader page of a file) has been extended to 24 bits, the top 8 of which designate the logical disk unit number. Hence files may refer to other files on different disk units. When the IFS follows an FP, it maps the logical unit number into the "disk object" to be accessed by the TFS package whenever a page of that file is referenced. The low-order 16 bits are then used as a virtual DA on the selected disk unit in the normal fashion.

The Serial Number (SN) structure is also extended to contain the following attributes:

archived (DA replaced by tape numbers of two archive tapes)
others?

New files are created on the emptiest disk unit. Disks may be added to (but not subtracted from)

the configuration without reinitializing the file system.

Directory Structure

The IFS directory consists of a single large file called <System>IFS.Dir, organized as a B-tree. This is a Legal IFS File containing a collection of Directory Records (DRs) mapping name strings into File Pointers (FPs). An FP in turn contains the Serial Number (SN) and Disk Address (DA) of the leader page of the file.

Pathnames in the IFS are in the form "<directory>directory> ... >filename!version". Each filename entry in the directory includes its complete pathname, including punctuation, with upper- and lower-case alphabetics considered to be equivalent. All printing characters except '*' are permitted, and the length of the entire pathname is limited to 99 characters.

The appearance of multiple user directories (and subdirectories within those directories) is entirely an artifact of naming conventions. That is, the set of files belonging to user Taft are simply those whose names begin with "<Taft>", and files belonging to Taft's subdirectory "Memos" are those whose names begin with "<Taft>Memos>".

Of course, the IFS attaches a certain amount of semantic significance to pathname syntax. In particular, all pathnames must begin with "<*dir*>", where *dir* is a directory name already defined in the IFS (this restriction does not extend to subdirectories, however). The properties of B-trees permit efficient enumeration of all files belonging to a particular directory or sharing any initial substring. The version number must be in the range 1 to 65535. The ordering relationship among pathnames is by Ascii code up through the "!" preceding the version, with lower-case collating with the corresponding upper-case alphabetics. Versions are sorted in increasing numeric order.

The IFS maintains essentially the same notions of *users* and *directories* as does Tenex. A user is identified by a string, is authenticated by a password, and has a set of attributes defining access capabilities and other properties. A directory is a set of files all of whose names begin with a particular "<*dir*>" and which are treated together with respect to certain directory-wide properties (overall directory protection, default file protection, disk page limit, etc.) Each user has a directory called "<*user*>" to which he has owner access. Additionally, there are a number of "files-only" directories whose names do not correspond to any user, but to which users may gain access either by mentioning the directory in a complete pathname (protections permitting) or by "connecting" to the directory so as to gain temporary owner access to it.

Properties of both users and directories are maintained in directory information files (DIFs) whose names consist entirely of the string "<*dir*>!1". A DIF may be created only by a user with administrative capabilities, and has a protection such that the owner is permitted to read but not to modify it. The DIF includes the following information:

User properties:
Password for "login" or "connect" (encrypted)
User group membership
Special capabilities (administrator, wheel)
Other administrative information

Directory properties:
Directory protection
Default file protection
Disk page limit
Other attributes (files-only, ...)
The name of the user responsible for the directory (if files-only)

The directory entry record for a DIF contains some additional information (beyond the usual name string and FP). This information is kept here for efficiency reasons, and includes:

1. Copies of information in the DIF that must be referenced on every access to any file in the directory (directory protection, default file protection, disk page limit).

2. Information recomputable by enumerating all files in the directory (disk page usage).

File Protections

The IFS implements a straightforward extension of the Tenex protection facilities. This extension is intended only to overcome a serious shortcoming of the Tenex facilities, namely the inability to specify protections on a per-group basis. A fundamentally different protection mechanism may eventually be implemented in the Distributed File System.

Each user belongs to one or more user groups, as in Tenex. The IFS permits the definition of up to 62 user groups, and a user's group membership is represented by a 62-bit vector.

Each file has a file protection consisting of three 64-bit vectors defining read, write, and append permission to the file. The first 62 bits of each vector define the set of user groups that are permitted the specified type of access (read, write, or append). The last two bits indicate access permission by, respectively, the owner and all other users.

To determine whether a user has read access to a given file, the IFS first appends two bits to the user's 62-bit group membership vector. The "owner" bit is set to one if the user's "login" or "connect" name is the same as the file's directory name, otherwise to zero. The "all other users" bit is set to one. This vector is then bitwise anded with the file's read permission vector. If the result is nonzero, the user has read access to the file. Write and append permission are determined in a similar fashion.

A directory as a whole also has two types of protection: *create* permission and *connect* permission. A user granted create access to a directory is permitted to create new files in that directory. A user granted connect access is permitted to "connect" to the directory without specifying a password. Create and connect permission are represented by two 64-bit vectors in the same fashion as file protections. They are contained in the DIF, and copies of them are maintained in the directory entry record for the DIF as described previously. (Note that these directory protections are distinct from the protections of the Directory Information File itself.)

Also maintained in the DIF is the default file protection to be used when new files are created in the directory. The owner of a file is permitted to change the file's protection regardless of whether or not he has write access to it.

File protections are inherited from one version to the next.

Special Files and Initialization

The TFS requires that each disk have its own SysDir and DiskDescriptor files. The SysDir and DiskDescriptor files for logical units 0, 1, ... are stored as Legal IFS Files named in the <System> directory as DiskDescriptor.0, SysDir.0, DiskDescriptor.1, etc. They are looked up during initialization and their FPs stored in the respective disk objects so as to be accessible to the TFS. The file <System>IFS.Dir is also entered in SysDir so that the IFS initialization can discover it.

Swapping space for virtual memory and overlays is reserved in a contiguous region on logical unit 0, and is entered as file "IFS.Swap" in both SysDir and the IFS <System> directory.

Each disk contains a "home block" recording vital configuration information, such as:

Number of disks in the file system
Logical unit number of this disk
Date and time of file system creation (for consistency check)
Name of file system

The home block contains sufficient information to permit the software to discover the configuration when it is started up. It is entered as file IFS.Home in SysDir and as <System>IFS.Home.*u* in IFS.Dir (where *u* is the logical unit number).

A few other special files are entered in both SysDir and IFS.Dir. These are IFS.Errors, a file that maps error numbers to strings; IFS.Syms, the symbols for the currently-running system; and IFS.Ov, a description of the code overlays (useful to the VMemSpy program).

The file system is initialized in the following fashion:

1. A virgin Alto file system is created on each unit by means of the standard TFS

initialization procedures. This causes all pages to be marked deleted and the files SysDir and DiskDescriptor to be created.

2. A home block file is computed and written on each unit.

3. Files IFS.Dir and IFS.Swap are created in contiguous storage on logical unit 0 and entered into unit 0's SysDir. The overlays are copied from the IFS.Run file on the Diablo disk to IFS.Swap on the Trident. From this point on, it is possible to do swapping on the Trident. (1000 pages for IFS.Dir and 100 for IFS.Swap should be sufficient in a file system containing up to 8 Trident disks.)

4. The <System> directory information file is created and entered into IFS.Dir. All the special system files are then entered into the IFS <System> directory (note that their leader pages must be modified to turn them into Legal IFS Files).

A new unit is added to the file system as follows:

1. A virgin Alto file system is created on the new unit, as above.

2. A home block is written on the new unit, and the home blocks on all other units updated.

3. The new unit's SysDir, DiskDescriptor, and IFS.Home files are entered into <System>.

Inter-Office Memorandum

To	IFS Project	Date	June 29, 1979
From	Ed Taft	Location	PARC/CSL
Subject	IFS Directory Operations (version 1.21)	File	[Maxcl]<IFS>IFSDirOps.bravo

XEROX

This document describes the BCPL interface to the IFS directory modules. Familiarity with the IFS file structure is assumed; see the memo "IFS File Structure". This memo supercedes the previous "IFS File Operations" memo.

Organization

The directory package consists of the following modules:

IFSDirs.decl	Parameter and structure declarations used within the directory package and needed when calling many of its procedures.
IFSDirOpen.bcpl	Procedures to open and close IFS files and create streams to which normal Alto disk stream operations may be applied.
IFSDirDelRen.bcpl	Procedures to delete and rename IFS files.
IFSDirParse.bcpl	Procedures to parse IFS filenames and build File Descriptors (FDs).
IFSDirLookup.bcpl	Procedures to look up files in the IFS directory.
IFSDirUtil.bcpl	Miscellaneous utility procedures needed by the other modules.
IFSDirKeyb.bcpl	The 'compare key routine' passed to the B-Tree package.
IFSDirKeya.asm	The 'length routine' passed to the B-Tree package.
IFSDirAdmin.bcpl	Procedures to create and destroy user Directory Information Files (DIFs).
IFSDirCheck.bcpl	A procedure to check the consistency of the directory B-Tree.

These modules call the B-Tree package and make use of other facilities provided in the IFS environment. There is a total of about 9000 words of code (including the B-Tree package), divided into 12 overlays of less than 1024 words each.

The directory package provides facilities at several levels. At the highest level are procedures implementing functions analogous to those in the standard Alto directory package. For example, the IFSOpenFile procedure translates directly from a file name to a stream in a manner similar to the Alto OpenFile.

At lower levels, specific file operations are passed a handle called a File Descriptor (FD).

Procedures exist to translate a file name into an FD, to look up the FD in the directory, to open or close a file given its FD, and so on. An FD may designate multiple files (due to wildcard '*' characters appearing in the name), and a procedure exists to step the FD from one such file to the next.

Directory and File Locks

Since the IFS is providing a multiple-access service, mutual exclusion mechanisms are required to maintain consistency of shared data structures. These mechanisms are relatively automatic when the directory package is accessed at its highest level. At the lower levels, however, callers must be aware of the resources that are locked at any given time.

Two kinds of locks are implemented, file locks and directory locks. An open file is either read- or write-locked. A file may have multiple readers but only one writer at a time, and no readers if any writer or vice versa. (A file opened for both reading and writing by a single process is write-locked.) A file lock is set at the time the file is opened, deleted, or renamed, and the operation will fail if the lock cannot be set. The lock controls access both to the file itself and to its directory entry.

In addition to the locks on individual files, there is a lock controlling access to the directory itself (recall that the entire directory is a single B-Tree). Directory access conforms to the same 'readers and writers' discipline as does file access, but inability to set the directory lock immediately causes the process to wait rather than resulting in failure of the file operation. Note that the directory lock controls access only to the directory itself; operations on files that are individually locked may proceed without regard to the directory lock.

Since the directory is shared among all users, it is essential that a process lock it for as little time as possible. In particular, operations that can take arbitrarily long (such as deleting a file) should not be performed while keeping the directory locked. Most directory operations (lookup and update) are completed quickly. A process performing a potentially lengthy directory operation (such as enumerating it) is expected to check periodically for occurrences of lock conflicts (other processes waiting to use the directory) and to release and reacquire the lock when conflicts occur.

A Lock is a two-word structure defined in IFS.decl. Lock.count contains a positive read lock count or -1 to denote a write lock, and Lock.ctx contains a pointer to the context that last set the lock. All locks are manipulated by means of the Lock and Unlock procedures in IFSResUtilb.bcpl. They are called as follows:

Lock(lock, write [false], returnOnFail [false]) = true or false

> Attempts to set the specified lock (a write lock if *write* is true or a read lock if false or omitted), and returns true if successful. If *returnOnFail* is false or omitted, blocks until the lock can be set; if true, returns false if the lock cannot be set immediately.
>
> A process should not attempt to set the same lock multiple times without an intervening Unlock. An attempt is made to detect occurrences of this error, but the check is not foolproof.

Unlock(lock)

> Unlocks the specified lock, which must have been either read- or write-locked by the same process.

File and directory locks are ordinarily manipulated by higher-level procedures such as LockFile and LockDirFD, described later.

Data Structures

Most IFS data structures are not operated upon directly by programs calling the directory package, but rather are simply passed as parameters. However, an understanding of the contents and function of the major data structures is helpful.

IFS data structures are divided into two classes, *file system* and *runtime*. The data structures actually stored on the disk are defined in IFSFiles.decl and are documented in "IFS File Structure". These will not be further discussed here.

The runtime structure IFS (defined in IFS.decl) designates an active file system. The software is capable of dealing with multiple, independent file systems simultaneously. All file operations are performed relative to a particular file system denoted by an argument *fs*. The default is the *primary* file system primaryIFS, which must be on-line when IFS is started and is the one used for swapping. Other file systems may be mounted and dismounted while IFS is running.

The IFS structure contains configuration information (in particular, a table mapping logical unit numbers to physical disk drives) and directory information, including pointers to the B-Tree structure and Open File Table (OFT), the directory lock, and two other interesting items relating to directory access.

The IFS.dirVersion word is a count of modifications to the directory: it is incremented every time the directory is modified in any way. This is useful to programs that wish to re-validate the results of an earlier lookup when the directory has been unlocked since that lookup. If, after locking the directory, the program finds that dirVersion has not changed since the last lookup, then the lookup information is still valid; otherwise, the lookup must be repeated (a relatively expensive operation). This feature is used by the LookupFD procedure, described later.

The IFS.dirLockConflict word is set to true whenever a directory lock conflict occurs, i.e., when a process attempts to set the lock and cannot because it is already locked in a conflicting way. A program intending to keep the directory locked for a long time should, after locking it, set dirLockConflict to false, then periodically poll it and, when a conflict occurs, briefly relinquish the lock so as to give the conflicting process a chance to proceed.

The Open File Table (OFT) contains the locks for all open files. It is a hash table, keyed on the virtual disk addresses of the open files.

The UserInfo block (defined in IFSFiles.decl) contains the identity of and information about the user for whom file operations are being performed. The directory package uses this information in order to check access to files and to record the creator of new files. It is the responsibility of other parts of IFS to create the UserInfo block, check passwords, and so on.

Most procedures in the directory package assume they are running within the confines of a Rendezvous Socket Context (RSCtx, defined in IFSRS.decl), which contains a pointer to the appropriate UserInfo block. A special UserInfo block, pointed to by the static *system*, exists to permit privileged, internal file operations that bypass access checking.

All lower-level directory operations are passed a structure called a File Descriptor (FD), defined in IFSDirs.decl. An FD is originally created by CreateFD, which parses a file name and sets up some auxiliary lookup information, such as the actual version number as an integer (if one was specified) and the indices of the end of the directory name and the end of the name body. The FD carries with it information that remains fixed for the life of the FD, such as the file system and the lookup control parameter. As operations are performed on the FD, various parts of it are updated.

High-Level Operations

The following operations are similar to ones available in the Alto Operating System.

IFSOpenFile(name, lvErrorCode [], mode [modeRead], itemSize [charItem], lc [see below], fs [primaryIFS], dirName [connected]) = stream or 0

Opens an IFS file, translating directly from a name to a stream. If successful, returns an open stream upon which the standard Alto disk stream operations may be performed. If unsuccessful, stores an error code in *@lvErrorCode* and returns zero.

The *name* must be a BCPL string whose complete form is '*<dir>name!ver*', but in which the directory and version may be omitted (in which case they will be defaulted). The default directory is the BCPL string *dirName*, if supplied, or the connected directory obtained from the running context's UserInfo block otherwise. The default version is given as part of *lc* (see below). The version may also be one of '!H', '!L', or '!N' designating highest existing version, lowest existing version, or next higher version. If the lookup control permits it, wildcard '*'s may appear anywhere in the name to designate multiple files (discussed in more detail later).

The *mode* should be one of modeRead, modeWrite, modeReadWrite, or modeAppend. The first three modes are equivalent to the corresponding ksTypes in the Alto operating system, while modeAppend is equivalent to modeWrite except that the stream is initially positioned to end of file. (IFSOpenFile correctly checks the write or append protection of a file when it is opened; however, it is the caller's responsibility to prohibit overwriting existing parts of a file opened in append mode.)

The *itemSize* is one of charItem or wordItem, as in the Alto operating system.

The lookup control (*lc*) parameter contains several bits and fields controlling certain aspects of the directory lookup process. If the lcCreate bit is set, then IFSOpenFile may create a new file (protections permitting); otherwise, if the designated file does not exist an error will result. If the lcMultiple bit is set, the name is permitted to designate multiple files by means of '*'; otherwise, occurrence of '*' in the name will cause an error. The remainder of the lookup control word is a default version control specification, to be used in the absence of an explicit version number in the name. This may be one of:

lcVHighest	highest existing version
lcVNext	next higher version (highest+1)
lcVLowest	lowest existing version
lcVAll	all versions (same as '!*')

The default lookup control specification depends on the mode in which the file is being opened, as follows:

modeRead	lcVHighest
modeWrite	lcCreate+lcVNext
modeReadWrite	lcCreate+lcVHighest
modeAppend	lcCreate+lcVHighest

If the name contains '*'s (which are accepted only if the lookup control includes lcMultiple), then the first file whose name matches the pattern is opened. It is expected that the caller will retain the FD and step it through all the other files matching the pattern, using the NextFD and OpenIFSStream primitives described later.

Closes(stream)

Performs the normal actions of cleaning up and destroying the stream, and also closes (i.e., unlocks) the file and destroys the FD.

IFSDeleteFile(name, lvErrorCode [], lc [lcVLowest], fs [primaryIFS], dirName [connected]) = true or false

Deletes the specified file, returning true if successful and false if unsuccessful. The parameters are interpreted as for IFSOpenFile. The name may not designate multiple files. The user must have write access to the file.

IFSDeleteOldVersions(name, lvErrorCode [], fs [primaryIFS], dirName [connected]) = true or false

Deletes all but the highest-numbered version of all files designated by *name*, which may include '*'s but must not have an explicit version number. Returns true normally and false if no file by that name exists or any of the delete operations fails; an error code for the last such failure is stored in @lvErrorCode.

IFSRenameFile(oldName, newName, lvErrorCode [], lc [lcVHighest], fs [primaryIFS], oldDirName [connected], newDirName [connected]) = true or false

Renames the file *oldName* to be *newName*, returning true if successful and false if unsuccessful. The old file must exist and the new file must not exist. The *lc* parameter applies to *oldName*; the lookup control used for *newName* is lcCreate+lcVNext. The user must have write access to the old file and create access to the user directory in which the renamed file will reside.

Lower-Level Directory Operations

CreateFD(name, lc, lvErrorCode [], fs [primaryIFS], dirName [connected]) = fd or 0

Parses the name and constructs an FD, returning the FD if successful and zero if unsuccessful. The only possible errors are syntax errors in the name; this procedure makes no references to the directory.

CreateFD sets the fs, lc, lenDirString, lenSubDirString, lenBodyString, and version fields in the FD structure. A skeleton Directory Record (DR, defined in IFSFiles.decl) is constructed and saved in the dr pointer; this record is of drTypeNormal and contains a copy of the name string, with an appropriate version number appended if appropriate (0 for lcVLowest and 65535 for lcVHighest or lcVNext).

If the lookup control includes lcMultiple and the name contains '*'s (or the version control is lcVAll and no version is specified in the name), a template is constructed and stored in the template field in the FD for later pattern matches, the index of the first '*' is stored in the iFirstStar field, and the name in the DR is truncated at that point for use as a starting key by NextFD.

All remaining fields in the FD are zeroed. In particular, the lookupStatus field is set to lsNoLookup, indicating that this FD has not yet been looked up in the directory.

DestroyFD(fd) = 0

Destroys the FD (and the DR and template pointed to by it, if any). Zero is returned so as to permit use in contexts such as:

fd = DestroyFD(fd)

LookupFD(fd, write [false]) = 0 or error code

Looks up the file described by *fd*, applying the lookup control parameters (already stored in the FD) as appropriate. If the file already exists, replaces the DR pointed to by the FD with a copy of the actual entry that was found (including its type, length, and FP). If the file does not exist but lcCreate is set in the lookup control, generates a complete DR (except for the FP) which may be used when creating the file.

Returns zero if successful and an error code if unsuccessful. Failure to find the file in the directory is considered an error only if lcCreate is not set or directory '<*dir*>' does not exist.

In the successful case, the lookupStatus field in the FD is set to one of the following:

lsNonexistent	The file does not exist, and no other version of that file exists either. In order to create the file, one must use directory-default file properties.
lsOtherVersion	The file doesn't exist, but another version of the file does exist. The FP of that file is stored in the DR to permit one to read its leader page and obtain is properties. (This facilitates the inheriting of properties from one version to the next.)
lsExists	The file exists, and a copy of its actual directory entry record is stored in the DR.

If the FD designates multiple files and this is the first time LookupFD has been called, LookupFD calls NextFD to find the first directory entry matching the pattern in the FD's template. If no such file is found, an error is returned.

LookupFD normally assumes that the directory is *not* locked at the time of the call. It sets a write lock if *write* is true and a read lock if false or omitted, and returns with the lock set *whether or not the lookup is successful.* One may call LookupFD with the directory already locked by passing a *write* argument of *dontLock.*

At the time of the return from LookupFD, and for as long as the directory remains locked, the FD (in particular, the lookupStatus) is *valid,* i.e., it accurately reflects the state of the directory. Hence one may immediately perform operations such as opening the file or modifying the directory entry.

However, once the directory has become unlocked, the FD is no longer valid, since some other process could change the directory at any time. In this case, before making use of the information in the FD one must *revalidate* it by calling LookupFD again, and one must be prepared for the possibility that the lookupStatus may change or even that the new call will fail despite the previous call having succeeded. (The revalidation operation is very cheap if the directory has not actually changed since the last LookupFD on the same FD.)

LookupIFSFile(name, lc, lvErrorCode [], fs [primaryIFS], dirName [connected]) = fd or 0

Combines the actions of CreateFD and LookupFD, returning an FD if successful and zero if unsuccessful. Unlike LookupFD, LookupIFSFile returns with the directory locked *only if it is successful.* The directory is write-locked if *lc* includes lcCreate and read-locked otherwise.

NextFD(fd, write [false]) = true or false

If *fd* designates multiple files, finds the next file matching the pattern and replaces the FD's DR with its directory record. If such a file is found, sets the lookupStatus to lsExists and returns true. If no such file is found, or *fd* does not designate multiple files, returns false.

NextFD assumes that the directory is *not* locked at the time of the call. It sets a write lock if *write* is true and a read lock if false or omitted, and returns with the lock set *whether or not the lookup is successful.*

LockDirFD(fd, write [false])

Locks the directory referenced by *fd.* Sets a write lock if *write* is true and a read lock otherwise. If a lock conflict occurs, sets the IFS.dirLockConflict flag and waits until the directory becomes unlocked.

UnlockDirFD(fd)

Unlocks the directory referenced by *fd.*

ModifyDirFD(fd)

Declares the directory referenced by *fd* to have been modified, by incrementing its version number. This should be done whenever the directory is modified (i.e., an entry is created, deleted, or updated). The directory must be write-locked at the time of the call.

Lower-Level File Operations

OpenIFSFile(fd, mode) = 0 or error code

Opens the file designated by *fd,* which must previously have been validated by a successful call of LookupFD or NextFD. Returns zero if successful and an error code if unsuccessful.

The directory must be *locked* by the caller (presumably as a result of calling LookupFD or NextFD). A write lock is required if OpenIFSFile is to create a new file (lookupStatus ne lsExists). OpenIFSFile returns with the directory *unlocked* regardless of whether or not it is successful.

OpenIFSFile checks protections and allocations as appropriate, creates the file if necessary, and attempts to read- or write-lock the file depending on the mode. If any of these operations fails, an appropriate error code is returned and the file is not locked.

CreateIFSStream(fd, itemSize) = stream or 0

Creates and returns a stream for an open file designated by *fd*. Returns zero if unsuccessful (the only likely cause of failure is a disk error in the leader page). *fd* is saved in ST.par1, which must not be clobbered by anyone (par2 and par3 are ok to use, however). Positions the stream to end of file if the file was opened with modeAppend.

OpenIFSStream(fd, lvErrorCode [], mode [modeRead], itemSize [charItem]) = stream or 0

Combines the actions of OpenIFSFile and CreateIFSStream, returning the stream if successful and zero if unsuccessful. All comments related to directory locking in OpenIFSFile apply here. If any operation fails, an error code is stored in *@lvErrorCode* and the file is not locked.

StreamsFD(stream) = fd

Returns the FD designating the file associated with the open stream.

CloseIFSFile(fd, dPages [0])

Closes (i.e., unlocks) the open file designated by *fd*, but does not destroy the FD. If *dPages* is supplied and nonzero, then the disk page utilization in the user's Directory Information File (DIF) is updated by the amount *dPages*, which may be positive or negative. *dPages* should be the amount by which the file's size changed while it was open.

The directory should *not* be locked at the time of the call and is not locked when CloseIFSFile returns.

CloseIFSStream(stream) = fd

Closes an open file given its associated stream handle, and destroys the stream, but does not destroy the FD. This procedure calls CloseIFSFile internally and takes care of the dPages computation. The FD is returned for convenience in enumerating multiple files. The Closes stream operation is identical to CloseIFSStream except that it also destroys the FD.

DeleteFileFromFD(fd) = 0 or error code

Deletes the file designated by *fd*, returning zero if successful and an error code if unsuccessful. This procedure deletes both the directory entry and the file itself. It differs from IFSDeleteFile in that it accepts an FD rather than a name and does not destroy the FD, so it is useful when deleting multiple files.

The directory must be either read- or write-locked at the time of the call (presumably by the validating LookupFD or NextFD) and is unlocked before the return whether or not the operation is successful.

ChangeFileProtection(fd, mask, value) = 0 or error code

Changes the protection of the file designated by *fd* according to *mask* and *value*, which are FileProt structures (see IfsFiles.decl). Bits corresponding to ones in *mask* are set to the new values in *value*. Bits corresponding to zeroes in *mask* are unchanged.

The caller must either be the owner of the file or have write access to it. The directory must be locked at the time of the call and remains locked upon return.

TransferLeaderPage(fd, buffer, write [false])

Transfers the leader page of the file designated by *fd* to or from the supplied page-size buffer. The page is written if *write* is true and read otherwise. The file need not be open, but if it isn't the directory must be locked and the FD must have been validated by LookupFD or NextFD.

Directory Administration Operations

ReadDIF(name, fs [primaryIFS], lvErrorCode []) = dif or 0

Reads the Directory Information File (DIF) for the supplied directory name, and returns a DIF structure (see IFSFiles.decl) which the caller must free when done with it. The initial DIFRec portion of the DIF is copied from the cached information in the DIF's directory entry rather than from the file itself so as to obtain the up-to-date value for the disk page usage. The caller must have read access to the DIF (which is ordinarily protected against all users except the owner).

WriteDIF(name, dif, fs [primaryIFS]) = 0 or error code

Creates or updates the DIF for the supplied directory name. *dif* must point to a completely filled-in DIF structure. This operation includes updating the information cached in the DIF's directory entry from the initial DIFRec portion of the DIF. The caller must have 'wheel' capability.

WheelWriteDIF(name, dif, fs [primaryIFS]) = 0 or error code

Performs the same operations as WriteDIF except that it pretends that the caller is a wheel and thereby bypasses access checks. It is the caller's responsibility to ensure that the operation is reasonable.

CreateUser(name, password, diskLimit [1000], owner [0], capabilities [0], worldRead [false], fs [primaryIFS]) = 0 or error code

Creates a new user directory (i.e., a DIF) with the parameters supplied and the remaining parameters set to default values. Returns zero if successful and an error code if unsuccessful. If *owner* is nonzero, a files-only directory is created and *owner* is taken to be a BCPL string specifying the directory's owner. The default file protection is set to give read access to the world if *worldRead* is true. The caller must have 'wheel' capability.

This procedure is useful primarily during file system creation for establishing the essential built-in directories (e.g., <System>). It provides means for setting only a subset of all possible directory parameters.

DestroyUser(name, fs [primaryIFS]) = 0 or error code

Destroys the named user directory, returning zero if successful and an error code if unsuccessful. All files whose names begin with '<*name*>' are destroyed, including the DIF. Since the files are deleted using normal access methods, the caller should be prepared to retry the call after errors such as ecFileBusy. The caller must have 'wheel' capability.

Inter-Office Memorandum

To	IFS Project	Date	August 6, 1978
From	David Boggs	Location	Coyote Hill
Subject	IFS debugging and tuning aids	Organization	Parc

XEROX

Filed on: <Ifs>IFSDebAids.bravo

This memo describes several programs which are used to debug and tune an Interim File System.

ListOV

ListOV reads the .Syms file produced by BLDR and outputs a text file listing the size of each overlay. Its primary function is to help partition code into swaping unit sized overlays. It is invoked by typing::

>ListOV <Subsytem-name>

The extension on <subsystem-name> is stripped if present and '.syms' is appended. The output text file is called 'Subsytem.ov'. The overlay size reported is the number of words which will be swapped in by the overlay package on an overlay fault -- the number of code words plus the number of relocation pair words plus one. The format of the size information is octal (decimal).

VMemSpy

VMemSpy spies on an IFS from another Alto and displays the virtual memory page currently resident in each real memory page. Its purpose is to show the dynamic utiization of real memory by the virtual memory. It is invoked by typing:

>VMemSpy <Host name> <Symbol filename>

<Host-name> is a Pup internetwork name and <symbol filename> is the .Syms file for the spyee (usually IFS.Syms).

VMemSpy obtains its information through a level 1 PUP connection to a spy process in the IFS. Periodically VMemSpy requests the spy process to send a snapshot of the virtual memory tables which it then uses to paint a picture of main memory on the Alto display. Pages which are locked in memory are displayed with a black background; pages which can be swapped out or overwritten are displayed with a white background. Dirty pages (i.e. pages which must be written back on the disk before reuse) are displayed with a gray patch to the right of the real address. Each time a reply is received, the cursor pattern is flipped.

VMemSpy displays the following strings in a page:

String	Page Contents
<Overlay name>	an overlay
DiskDescriptor	a disk descriptor file page (disk bit table)
VFile	a file page accessed through VMem (usually a BTree)
Snarfed	unknown (someone borrowed it from VMem)

Free	nothing

The VMemSpy causes only a slight perturbance to the IFS under scrutiny so it may be used to observe a service system. Since VMemspy and the spy process inside the IFS communicate using a connectionless protocol, several VMemSpys may spy on the same IFS.

AllocSpy

AllocSpy spies on an IFS from another Alto and records information about each call on the free storage manager. Its purpose is to detect bugs which lose free storage and to show the static and dynamic utilization of free storage. Results are written to the file "AllocSpy.log". It is invoked by typing:

>AllocSpy <Host name> <Symbol filename>

<Host-name> is a Pup internetwork name and <symbol-filename> is the .Syms file for the spyee (usually IFS.Syms).

AllocSpy works by making a BSP connection to a spy process in the IFS. The spy replaces SysZone with a dummy which allows it to trap all calls on Allocate and Free. When a call on Allocate happens, the spy gets control and sends the following information to AllocSpy which saves it in a hash table:

size of the requested block
location of the resulting block
overlay relative PC and overlay number of the last three frames.

When a call on Free happens, the spy gets control and sends the location of the block being freed. AllocSpy deletes the block from its table. Each time a block is allocated AllocSpy displays an "A" and each time a block is freed it displays an "F". When an allocate event is received and the block is already in the table or when a free event is received and the block is not in the table an "←" is displayed after the "A" or "F".

There are two keyboard commands:

M [Map] causes a snapshot of the current state of all free storage to be appended to the log. This is a rather lengthy process so when it is complete AllocSpy displays a ".". AllocSpy automatically requests a map when it opens the connection.

Q [Quit] cleans up and returns to the Exec. All allocate events currently in the hash table are interpreted and output to the log file as 'orphaned blocks'.

AllocSpy should be run on a test system after any significant changes have been made. It significantly degrades the performance of an IFS so it should not be run on a service system unless absolutely necessary. AllocSpy and the IFS should be connected via a high bandwith path since calls on Allocate and Fee can only proceed as fast as the spy can push the information through the byte stream.

Inter-Office Memorandum

To	IFS Project	Date	October 24, 1977
From	David Boggs	Location	PARC/CSL
Subject	IFS Scavenger	File	[Maxc1]<IFS>IfsScavDesign.bravo

XEROX

This memo outlines the design of the scavenger for the Interim File System (IFS). The IFS file structure is described in a separate memo. A scavenger is particularly important for the IFS since the directory mechanism does not always maintain a consistant structure on the disk.

The scavenging process is divided into two passes. Pass1 verifies the file structures, and pass2 verifies the directory structures. Pass1 is run on all packs in the IFS, and then pass2 is run on logical unit 0 (which contains the IFS directory). Scavenger data structures are maintained on a scratch disk since they can grow quite large.

Pass1 - File structure verification

Pass1 builds two data structures, the Page Link Map (PLM) and the Leader Page Table (LPT). The PLM is used during pass1 and then discarded, and the LPT is used during pass2. The objective of pass1 is to insure that the disk is a well formed Alto file system and that some IFS-specific files are present. The output of pass1 is the LPT, which is a list of all legal files in the system. Since files in an IFS can't cross disks, pass1 can treat each disk separately, reusing the PLM file. For each pack in the system, the following steps are taken:

1) For each page of the pack, an entry is made in the PLM containing the page's forward and back links, fileID, page number, and numChars. Whenever the scan encounters a leader page, its FP, hintLastPageFa, and filenames (both ifs and tfs version) are checked and appended to the LPT.

A PLM entry is 8 words long. The table below gives the size of the PLM (in scratch disk pages) for all combinations of scratch disks and disks being scavenged.

	T-300	T-80
Trident	512	287
Diablo	2048	1147

The average length of an LPT entry is 25 words. The table below gives the size of the LPT (in scratch disk pages) for representative numbers of files.

	5000	10000	25000
Trident	123	244	611
Diablo	492	977	2442

Things start getting tight around 10000 files on a T-300 using a Diablo as the scratch disk.

2) Starting from the list of leader pages, the PLM is scanned to verify the consistancy of pointers and serial numbers within each file. Bad files are deleted, last page hints are verified, accessible pages are so marked.

3) The PLM is enumerated, and all inaccessible pages are made free. A bit table is built on a scratch file. Any labels that step 2 didn't like are changed. The bad page list for the pack is updated. Any leader pages that steps 1 or 2 didn't like are changed.

4) SysDir and DiskDescriptor are rebuilt from scratch. Multiple directories are not supported.

5) This step is only performed if the pack is part of an IFS. It verifies IFS.home, sets the unit number in the IFPs of all LPT entries for this pack, and verifies the existance (although not the contents) of all critical IFS files.

Pass2 - Directory Structure Verification

The objective of pass two is to insure that the IFS.dir is a well formed B-Tree and that all files are accessible from it. Pass two consists of the following steps:

1) The LPT is sorted by filename, using the same comparison algorithm that the IFS uses to lookup files in IFS.dir.

2) The B-Tree structure of IFS.dir is verified. If the tree is damaged beyond the ability of this step to fix, it is initialized to empty.

3) IFS.dir and the LPT are enumerated in parallel. The LPT is the truth about what should be in the directory, so any disagreements are resolved by changing the tree.

The algorithm is

LPT entry > Tree entry:	delete tree entry, read next tree entry
LPT entry = Tree entry:	read next LPT entry, read next tree entry
LPT entry < Tree entry:	insert LPT entry, read next LPT entry

When a Directory Information File (DIF) entry is encountered or inserted into IFS.dir, the information cached there from the DIF is verified, and the recomputable information for the last DIF entry is verified.

Recomputable information currently consists of the number of pages in use. The directory group membership, default file protection, and page allocation are cached in a DIF entry, but the truth is kept in the DIF file.

Inter-Office Memorandum

To	IFS Project	Date	November 2, 1977
From	David Boggs	Location	PARC/CSL
Subject	IFS Scavenger Operation	File	[Maxc1]<IFS>IfsScavOp.bravo

XEROX

This memo describes the operation of the Interim File System Scavenger. The design is outlined in a separate memo. The scavenger reads every page in the file system, makes sure that every file is well-formed, and checks for consistency between files and directories. Since an Interim file system is (nearly) a superset of an Alto file system, this scavenger can also repair non-IFS Trident file systems.

When to Run the Scavenger

File system problems can be divided into two classes: 1) directory or bit table inconsistencies or a few malformed files, and 2) massive losses of data. The scavenger is intended to avoid the time-consuming process of reloading the file system from backup when the problems are minor. If a drive cuts loose and obliterates a large part of a pack, then you will probably lose fewer files by reloading from backup.

The most common causes of damage to an IFS are software bugs and hardware glitches which cause the system to stop in an unclean way. There is only one safe way to stop IFS: log in and issue the privledged HALT command. Any other method (including <Ctrl-K> from Swat and <Shift-Swat>) can potentially damage the file system. There are consistancy checks scattered throughout the system which should detect problems and call Swat before extensive damage is done. If this happens, or you are suspicious, then run the scavenger.

Calling the Scavenger

The scavenger is invoked by the command

IfsScavenger/switches

where the switches control the operation of the system in various ways. Switches defined at present are:

- /D Debug mode (various non-fatal errors call Swat rather than just continuing on).
- /A Allocator debug (every call to the storage allocator causes a very thorough consistency check to be invoked. This slows down operation of the system considerably.)
- /B Enable calls to Block within the disk driver, thereby permitting more overlapping of computation with disk transfers.
- /U Enable the microcode version of the Bcpl runtime.

As you can see, these switches are only useful for debugging; in normal operation none of them should be used.

After considerable churning on the Diablo, the scavenger will announce itself along with its release date and then wait for commands. The herald is an asterisk. The standard editing characters, command recognition features and help facility (via "?") are available.

Normal Operation

The scavenger scans each pack in the file system and then goes back and works some more on the pack containing logical unit 0. Although the order of scanning is unimportant, if logical unit 0 is still online when all of the packs have been scanned, you will not have to remount it, so I recommend that you scan it last. In the example below, what you type is underlined.

*Scavenge
Which drive shall I use for scratch? DP0 or TPm (see below)
Scan pack on drive TPn ($0 \le n \le 7$)

When it is done with that pack, if there are more packs in the file system it asks you to mount the next pack:

Scan pack on drive TPn

When it has scanned all of the packs, if it is not sure which drive has logical unit 0 it asks:

Which drive has logical unit 0? TPn

Finally, it says:

Scavenge complete
*Quit

Print IfsScavenger.log (see below), restart IFS, restore damaged files from backup, and notify owners of lost files not protected by backup.

Scratch Disks

The scavenger builds some large data structures which it must keep on some disk. It can use a Diablo or a Trident. The scavenger runs about twice as fast using a Trident.

If you have only one Trident drive, then you must use the Diablo for the scratch disk. This disk should be very empty. The number of free pages required is a function of the size of the largest disk and the number of files in the IFS. The table below estimates the minimum number of free pages needed to scavenge representative configurations.

largest disk	1000 files	5000 files	10000 files	25000 files
T-80	1250	1650	2150	3600
T-300	2150	2550	3050	4500

I recommend that you keep a disk with just the following files:

Sys.boot	Sys.errors	Sys.syms	SysFont.al
Swat	Swatee	Executive.run	Ftp.run
Ifs.run	Ifs.syms	Ifs.errors	Ifs.ov
Empress.run	Fonts.widths	Tfu.run Triex.run	
IfsScavenger.run	IfsScavenger.syms	Ftp.run	

If you have more than one Trident drive, I recommend that you use one of them for the scratch disk. The scavenger only needs one IFS pack online at a time, so you can do this even if you use all of the drives when running the file system. Depending on the size of the IFS, the scavenger will need between a few hundred and a few thousand pages on the scratch disk (divide the numbers in the table for the Diablo by 4). Any Trident disk with enough pages will do, though things will go faster if the pages can be allocated contiguously. I recommend that you use a freshly erased scratch

disk.

Scavenger Log

The scavenger appends display output to a typescript file (IfsScavenger.log) on the Diablo disk. You can use this information to notify the owners of files to which the scavenger does something drastic. During normal operation, the scavenger will display some messages telling what it is doing and summarizing statistics about the file system. A section at the end of this memo explains the messages.

Other Commands

There are a number of commands in addition to Scavenge; most have to do with debugging the scavenger.

Quit
: Returns control to the Alto Executive.

Pass1
: Scans a pack and makes it into a well-formed Alto filesystem. The dialog goes as follows:

```
*Pass1
Which drive shall I use for scratch? TPm
Scan pack on drive TPn
Is this an IFS pack? [Confirm] Yes
Which file system (0-2)? 0
. . .
May I alter your disk? [Confirm] Yes
```

: Alert readers will have noticed the similarity between this dialog and that for the Scavenge command. The Scavenge command is implemented by repeatedly calling Pass1 until all of the packs in the IFS have been scanned and then calling Pass2. The Scavenge command tells Pass1 that it is working on an IFS, that it's OK to make changes, and that if it is a T-300 then there are two Alto filesystems (0 and 1) on the pack.

Pass2
: Verifies the health of the directory B-Tree, and then verifies that all of the files discovered during the previous scans are listed in it, and no others. If Pass2 is invoked without previously running Pass1, it assumes that you ran Pass1 during a previous invocation of the scavenger. In this case it asks you which disk to use for scratch, and assumes that the proper scratch files are out there.

Debug
: This command turns on debug mode. More debugging information is output to the display, including some non-fatal error messages such as soft disk errors. The scavenger will pause after each phase has been swapped in so that breakpoints can be set. Typing any character proceeds it.

DiskEditor
: Invokes a simple disk editor with a DDT-style command syntax. The editor is described in more detail below.

DumpLPT
: Dumps (converting to text format) the contents of the leader page table (one of the scratch files) into a file on the Diablo. The dialog goes as follows:

```
*DumpLPT
```

Which disk is it on? TPn (see below)
What shall I call the output file on the Diablo? foo
Do you want just page usage info? [Confirm] No

If you have previously specified a scratch disk, then DumpLPT assumes that the LPT is to be found there, otherwise it asks. The text file produced by this command lists each file, its fp and selected information from the DIFRec if present. If you answer 'yes' to the last question, then only the number of pages used and the page usage limit for each directory is output. The LPT is deleted by pass2 unless the debug flag is set.

DumpTree
Dumps (converting to text format) the contents of the IFS directory B-Tree into a file on the Diablo. The dialog is very similar to DumpLPT, and the description for that command applies here too except that you are always asked for the disk.

EditHome
Sets a flag which allows you to edit the information in Ifs.Home when it comes under scrutiny during Pass1.

InitTree
Sets a flag which causes the scavenger to ignore the contents of the directory B-Tree and instead recreate it from scratch. This takes much longer.

Scavenging Non-IFS Trident Packs

To scavenge a non-IFS Trident pack, just run Pass1 on it and reply 'No' when it asks if it is an IFS pack.

Disk Editor

The scavenger contains a simple disk editor with a DDT-style command syntax. I wrote it so that I could damage a file system in controlled ways and test the scavenger's ability to fix the damage. It has turned out to be a useful tool in its own right. To start the disk editor type:

*DiskEditor
What disk would you like to edit? TPn

When the editor is running, the normal small display is replaced with a large one. The top level commands are (all numbers are octal):

<number>/
close the current page and open the page whose virtual disk address (vda) is equal to <number>. If no <number> is typed, the number last printed is used. The display looks like:

1/ fid 200000144;1 pn 0 nc 4000 177777 <-> 2

1/ means vda 1 is open.
fid 200000144;1 is the serial number;version number.
pn 0 means page number 0 of the file.
nc 4000 means this page contains 4000 bytes (it's full).
177777 <-> 2 means the back link is EOF and the next page is 2.

Typing just '/' follows the forward link in the currently open page.

\
close the currently open page and open the page pointed to by its back pointer. A number before the '\' is illegal, as is typing '\' when no page is open.

lineFeed
close the currently open page and open the one with the next higher vda. This sweeps the disk in ascending virtual disk address order (until your finger gets tired).

↑
close the currently open page and open the one with the next lower vda. This sweeps the disk in descending virtual disk address order.

return
close the currently open page.

Q
quit the disk editor (after confirming) and return to the scavenger's top level command scanner.

L
enter an editor for the Label record of the currently open page.

D
enter an editor for the Data record of the currently open page.

When editing a Label or Data record, the following commands are available:

<number>/
close the currently open cell and open cell <number> in the record. If no number was typed, the last number displayed is used. The display looks like:

```
1 = SN1/      40502                (if it is a label) or
1/ 40502      101 102 AB     (if it is a data record).
```

1/40502 means cell 1 of the record is open and contains 40502.
101 102 is 40502 displayed as bytes.
AB is 101 102 displayed as ascii characters.

Typing just '/' now would try to open cell 40502 which is out of range, so the screen would flash.

<number>return
If <number> was typed, store it in the currently open cell. Close the currently open cell.

<number>lineFeed
If <number> was typed, store it in the currently open cell. Close the currently open cell and open the next cell.

<number>↑
If <number> was typed, store it in the currently open cell. Close the currently open cell and open the cell before it.

Q
return control to the page editor. If you changed the record, you will be asked to confirm rewriting the changed record back onto the disk. If the record is rewritten, the page is closed.

Reporting Scavenger Bugs

Assuming that the hardware is in good health, the scavenger should be bullet-proof: no matter how badly mangled the file system is, the scavenger should not go into Swat. If it does, or you believe that the scavenger did the wrong thing, please take the following steps:

1) If you landed in Swat, make a sysout file (type <Ctrl-L> and supply a descriptive filename with extension .Swat), and then boot the machine.

2) Save IfsScavenger.log.

3) Get in touch with me.

As you can tell from the amount of debugging machinery in the scavenger, I have no illusions about it being correct.

An Annotated TypeScript

What follows is the typescript from scavenging an IFS that was in good health. Commentary is in a small font to distinguish it from the typescript; what I typed is underlined. The numbers in square brackets tell what module is generating the message. The format is [Pass-Phase]. Places where the scavenger would have paused if the debug flag was set are marked with an asterisk.

Ifs Scavenger of October 19, 1977
*Scavenge
Which drive shall I use for scratch? tp0 Use the freshly erased pack on drive 0 for scratch.
Scan pack on drive tp1 A single-pack IFS, mounted on drive 1 (a T-80).
[1-1] * Reads each page on the disk and builds the page link map (PLM) and
[1-1] time = 3:13 the leader page table (LPT). File name syntax is checked.
[1-1] files = 1038 The disk has this many pages which look like leader pages.
[1-2] * The forward link in each leader page is followed checking the file structure.
[1-2] time = 0:26. Every page which is part of a file is marked accessible.
[1-2] 11678 pages used out of 36674 Last page FA hints are checked.
[1-3] *
[1-3] PLM The PLM is enumerated and all inaccessible pages are made free.
[1-3] time = 0:30. A bit map is built. Damaged files are repaired or deleted.
[1-3] BPL The list of incorrigible pages is updated.
[1-3] LPT Leader pages which need work are rewritten.
[1-3] time = 0:2.
[1-4] *
[1-4] SysDir SysDir is rebuilt from scratch.
[1-4] time = 0:2.
[1-4] DiskDescriptor DiskDescriptor is rebuilt from scratch, using the bit table from [1-3].
[1-5] *
File system type: Primary Ifs.home is verified.
File system name: Test If the EditIIome flag is set,
Number of units: 1 you would be able to edit these 4 items.
Logical unit number: 0
[1-5] LPT The LPT is scanned looking for special system files
[1-5] time = 0:10. which are listed in the Special File Table, SFT.
[1-5] SFT If any files in the SFT were not found, they are created.
Pass1 complete
[2-1] *
[2-1] Time = 0:35. The LPT is sorted in directory order.
[2-1] Number of files = 1038.
[2-1] SortZone size = 19456. words.
[2-2] *
[2-2] PostOrder The directory B-Tree is traversed checking its structure.
[2-2] Time = 0:4.
[2-2] 2 levels, 41 pages allocated, 22 used.
[2-2] Free List The list of allocated but unused tree pages is rebuilt from scratch.
[2-2] Time = 0:2.
[2-3] * The LPT and the Tree are enumerated in parallel.
[2-3] Time = 2:51 The Tree is made to agree with the LPT.
Pass2 complete

*quit

Revision History

October 24, 1977
First release.

November 1, 1977
Added the abiltiy to scavenger multiple Alto file systems on a single T-300.

Inter-Office Memorandum

To	Laurel Group	Date	November 30, 1978
From	Ed Taft	Location	Palo Alto
Subject	Interim IFS Mail Forwarding (edition 2)	Organization	PARC/CSL

XEROX

Filed on: [Maxc1]<Taft>IFSForwarding.bravo

It has been proposed that we implement mail forwarding among IFSs, using the current mail transport software and protocols and a single name space. While the new Laurel transport mechanism, presently under development, will provide automatic forwarding within a distributed name space, we think it is desirable to provide an interim facility, for at least the following reasons:

1. Operationally and administratively, the current situation is becoming intolerable. We cannot cope with the present influx of new Alto users who need Maxc accounts just so they can have mailboxes. On the other hand, the new users need *some* way to send and receive messages between now and the time we complete the new Laurel transport mechanism.

   Indeed, one organization (WRC) has already found us to be sufficiently unresponsive to the needs of their new users that they have set up their own local mail system, using the IFS mail server. Since IFS-based mail systems can't communicate with each other or with Maxc, this is likely to lead to serious difficulties and confusion.

2. The new Laurel transport mechanism will be totally incompatible with the present one, and, in particular, will *not* make use of the existing Maxc and IFS mail servers. Furthermore, the new mail server will be written in Mesa and will therefore not be able to live in the same machine as an IFS. It is likely that many organizations will not immediately be in a position to devote another Alto to be a mail server.

3. Finally, the anticipated growth of the Alto user community will lead quickly to a situation in which we (CSL) come under pressure to make the new transport mechanism available as a service. This is a situation we are explicitly trying to avoid.

It is desirable that any interim arrangement involve as little work as possible, since such work will not contribute to the new Laurel transport mechanism. Implementation should be completed within one month, and its lifetime should be on the order of six months to a year.

Current situation

At present, Maxc and IFS have compatible Laurel mail servers. Each Maxc and IFS is capable of receiving messages directed to users whose mailboxes are located on that machine, and of retrieving the contents of those mailboxes.

Additionally, Maxc, but not IFS, has a *mail forwarding* capability. If a message arrives for a recipient whose mailbox doesn't exist on the local machine, the recipient name is looked up in a *mail forwarding data base*, which yields the name of the server believed to contain the recipient's

mailbox. The message is then queued for delivery by a mail-forwarding process that implements the user half of the Mail Transport Protocol.

The mail forwarding data base is maintained centrally on Maxc. Updating it involves having a privileged user edit a text file and recompile the data base. This is an inconvenience, but it is much less inconvenient than establishing and administering large numbers of individual user accounts.

Plan

Three different schemes for improving the current situation were proposed in the previous edition of this memo. They were considered on November 28 at a meeting of the combined ASD/CSL Laurel group. To help evaluate these proposals, I offered the following goals against which they should be measured.

a. Reduce the burden on the Maxc administrators by eliminating the need for non-Palo Alto users to have Maxc accounts.

b. Reduce the computing and file storage burden on Maxc by locating many mailboxes on IFSs.

c. Minimize the central control or coordination required to introduce or remove users.

d. Not require any substantial changes to user interfaces.

e. Require as few modifications as possible to existing software.

This plan is based on my recollection of the consensus of that meeting.

Overview of the design

Under this scheme, each mail recipient is associated with a home mail server, whose name is used as if it were a *naming authority* within the ultimate Laurel transport mechanism. When composing a message, a user must specify the names of non-local recipients in the form *user.host*. (This is similar to what users must do to send messages within the Arpanet. We use "." rather than "@" as a separator because of problems with encapsulating recipient names in messages forwarded to the Arpanet; see below.)

All Laurel users set their Laurel.Profile to send, retrieve, and authenticate on their home mail servers. Each mail server must accept recipient names of the form *user.host*, and must forward copies of such messages to the correct *host*. Note that this scheme does not require a mail forwarding data base. However, it does require that centrally-maintained distribution lists specify all members in the complete form *user.host*.

This proposal measures up to the goals as follows:

a. No central administration is required.

b. Maxc need forward only messages sent by users whose home mail server is Maxc, and need not maintain a mail forwarding data base (except for incoming Arpanet messages; see below).

c. No central name assignment is required. However, the *user.host* naming convention has some consequences affecting distribution lists and incoming Arpanet mail.

d. Users must know which recipients are non-local, and must know the location of each non-local recipient's mailbox. We can ease this by assigning organizational aliases to the mail server machines. (This has the consequence that all members of a given organization must have mailboxes on the same machine. This will probably require that we assign ficticious

organization names such as ASD-N, ASD-S, etc.)

e. Laurel, IFS, and Maxc must be changed.

1. Laurel must be changed to identify the sender of outgoing messages in the form *user.host* if there are *any* non-local recipients of the message. Only if all recipients are local can the *.host* be omitted. This is similar to the manner in which Laurel handles messages to the Arpanet now. Furthermore, the existing logic for handling Arpanet recipients must be changed to send such messages to *user@ArpaHost*.Maxc1 if the user's home mail server is not Maxc1 or Maxc2.

2. A mail forwarder must be added to IFS. This includes an MTP user (only partially implemented at present), code to manage outgoing queues of messages to other servers that might be down, and some means for returning undeliverable messages to their sender.

3. The Maxc mail server already knows how to forward messages according to this scheme. However, there are some technical problems with the current implementation of mail forwarding on Maxc. These are:

a. A separate file is created for each copy of each message to be forwarded to non-Maxc recipients. This is expensive in both space and time and could easily cause the Maxc <System> directory to overflow.

b. The mail forwarder is a single process that keeps no memory of which other mail servers are up and which are down. It could get badly clogged up as a result of an IFS going down, since it would have to time out each attempt to deliver each copy of each message destined for that IFS.

4. The ListMaker program (used to construct all the distribution lists from a common data base) must be modified to name *all* recipients in the form *user.host.*

5. The Maxc mail forwarding data base must include the names of those non-Maxc recipients who regularly receive messages from the Arpanet.

Laurel modifications

In the following, let *sender* be the name of a user running Laurel and composing or answering a message, and *home* be *sender*'s home mail server.

1. Laurel must recognize recipient names of the form *user.host.* When one or more of these appears in the header of a composed message, the "from" field must be set to *sender.home.* (Laurel may, but need not, strip off *.host* if *host* equals *home.* This may be desirable to eliminate unnecessary qualification, particularly when recipient names are taken from a distribution list, where names are always qualified.)

2. When answering an incoming message, Laurel must detect cases in which that message's "from" field is in the form *user.host*, where *host* is not equal to *home*, and interpret all unqualified names elsewhere in the header relative to *host.*

3. Laurel must detect recipient names of the form *user@ArpaHost* in the header of a composed message. If any of these appear, then the following operations must be performed:

a. In the message header, the "from" field must be set to *sender.home*@Parc-Maxc. (*sender*@Parc-Maxc is adequate if *home* is Maxc1. Note that this case is the reason for choosing a separator character different from "@": it is likely that many message systems on the Arpanet would incorrectly interpret names of the form *sender@home*@Parc-Maxc.)

b. Non-Arpanet recipients must be fully qualified as *user.host*. This is so that answers generated by the Arpanet recipients will be forwarded correctly when they come into Maxc from the Arpanet. (This operation can but need not be omitted if *host* is Maxc1.)

c. In the *property list* (but *not* in the header), Arpanet recipients must be specified as *user@ArpaHost*.Maxc1.

4. When answering an incoming message, Laurel must detect cases in which that message's "from" field is in the form *user@ArpaHost*, where *ArpaHost* is not Parc-Maxc, and interpret all unqualified (by "@") names elsewhere in the header relative to *ArpaHost*. (I believe this is what Laurel already does.) It should also strip off any occurrences of @Parc-Maxc in the recipient list. This is to prevent the intra-Xerox copies of the message from having to be forwarded via Maxc1 (by step 3c above).

IFS modifications

1. The mail server must accept recipient names of the form *name.host*, where *host* is a legal Pup host name. (The server should verify this before accepting the recipient name.) If *host* is the name of that mail server, it should strip off the *.host* and attempt to treat *name* as a local recipient name.

2. Either the mail server or the background mailer process must segregate the names of local recipients from those of non-local recipients, and must queue a copy of the message (along with the appropriate subset of the recipient list) for delivery to each different mail server *host*.

3. Queues of messages to each *host* must be maintained independently so that if one host is down, messages to other hosts aren't blocked.

4. If a message fails to be forwarded successfully, the mailer must deliver a return message to the sender (whose mailbox is local), enclosing a copy of the original message. Unsuccessful delivery is defined to be one of:

   a. Rejection of one or more recipient names by the remote mail server, indicated by receipt of a [Mailbox-Exception] response.

   b. Failure to establish contact with the remote mail server for some timeout interval (a few days), or repeated failure to deliver mail after apparent successful contact.

Maxc modifications

The functional changes required to Maxc are quite minor. However, some changes to the existing implementation are also required to prevent overflowing the directory on which mail is queued for forwarding.

1. The mail server must be modified to accept recipient names of the form *name.host* and *name@ArpaHost*.Maxc1. (At present, it accepts only *name@host* and *name@ArpaHost*.)

2. It must queue messages for each *host* in a manner similar to that described above in steps 2 and 3 for IFS. The present implementation of message queueing must be modified to eliminate the explosion of files created for each copy of each message to each non-local recipient.

3. The Mailbox program (called as a subroutine by a number of programs, including the Pup and Arpa mail servers and Sndmsg) must be modified to accept recipient names of the form *user.host*, where *host* is a legal Pup host name, and to return *user* and *host* separately. (It now accepts *user@host*.)

4. The ListMaker program must be changed to permit home mail server names to be associated with each user, and to generate distribution lists consisting of names of the form *user.host.*

Schedule and Manpower

Changes are required to Laurel, IFS, and Maxc. We have agreed informally that the changes to Laurel and to Maxc should be CSL's responsibility. The changes to IFS probably require about as much work as the Laurel and Maxc changes combined. Since Steve Butterfield has been responsible for development of the IFS mail server to date, it seems sensible to assign the IFS modifications to ASD.

ASD does not have any immediate need for an IFS mail forwarding capability for installation in probe sites, at least for the next year or so. Therefore, it may be difficult to justify Steve's devoting the next few weeks of his time on this project. On the other hand, if CSL makes the IFS modifications, Steve will ultimately have to spend some time getting the ASD and CSL versions of IFS to converge again.

This matter will require further negotiation.

It is important that the interim forwarding mechanism be implemented at once, since we expect it to be used for one year at most and hence its value decreases rapidly the longer we wait to start using it.

Inter-Office Memorandum

To	IFS Users	Date	April 24, 1982
From	Ed Taft and David Boggs	Location	Palo Alto
Subject	How to Use IFS (version 1.36)	Organization	PARC/CSL

XEROX

Filed on: [Maxc2]<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.

Information in this memo applies to all IFSs except where otherwise noted. To obtain a directory on your local IFS, or to find out about local operating practices, consult your IFS administrator.

This edition describes IFS version 1.36. User-visible changes since version 1.34 are as follows:

> IFS can be configured to use Grapevine for authentication and access control. This substantially changes the facilities for dealing with file protections; see section 3 for details.
>
> The Show System-Parameters command has been added.

1. 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 the Alto FTP program, or through other programs that use the FTP protocol such as the Mesa FileTool.

You should consult the FTP documentation in the *Alto User's Handbook*, the Alto Subsystems manual, or [Maxc2]<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; you must delete them manually. The Delete command, described below, has an option that facilitates this.

'*' 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 appearance of sub-directories is entirely an artifact of naming conventions; the file system actually keeps track of the complete file name for each file. '<' and '>' are ordinary characters, and '*' matches them the same as any other character. For this reason, if you use sub-directories at all, you will find them most useful if you employ them in a consistent fashion.

2. 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 or CTRL-A erases 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 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. If you terminate *new-filename* with ESC rather than RETURN, the body of *old-filename* (with directory and version stripped off) will be appended to *new-filename.*

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.

@ Change Attributes (of files) *file-designator*
@@ *sub-commands*

Changes certain attributes 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*
@@ Reset (all existing access)

Changes file protections, as described in section 3.

@@ Type *type*
@@ Byte-size *number*

Changes the type and byte size used for subsequent FTP retrievals of the files. *Type* may be 'Text', 'Binary', or 'Unspecified'. You should change these attributes only if you thoroughly understand their purpose and interpretation (see the FTP documentation in *Alto User's Handbook*).

@@ Backup
@@ No Backup

Enables or disables the automatic maintenance of backup copies of the files. Ordinarily, all files are backed up. Specifying 'No Backup' is appropriate only for very large files that are themselves duplicates of files stored elsewhere; you should use 'No Backup' only after consulting with your IFS's administrator.

All requested changes take effect when you type RETURN immediately after the '@@' prompt.

@ 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.

@@ Duplex

Specifies that the document is to be printed on both sides of the paper (if the printing server is capable of doing so).

@@ Password *password*

Causes the document not to be printed immediately but rather to be held by the printing server until you type *password* on the printing server's terminal. Only certain servers are capable of password-protected printing. Contact your local support staff for instructions on using this feature.

@@ 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.)

@@ 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 '*#*').

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

See section 3.

@ 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
@ Show System-Parameters

Prints out various operating statistics and parameters that are generally of interest only to IFS administrators. The Show System-Parameters command may be used to find out what services are enabled on the IFS, whether or not Grapevine is used for authentication and access control, etc.

3. Passwords, protections, and Grapevine

IFS has facilities for controlling access to a file server as a whole and to individual directories and files within the file server. A file server that is in communication with a Grapevine registration server can use Grapevine for authentication and access control. A file server that does not have access to Grapevine (or whose administrator has chosen not to enable the Grapevine facilities) maintains its own local data base for these purposes.

Since the IFS's behavior as seen by users is quite different in the two cases, we first describe the

case in which the IFS is using Grapevine for authentication and access control; then we present the additional facilities required to manipulate the IFS's local data base if Grapevine is not being used.

This description assumes that you understand the organization of Grapevine names and groups and are familiar with the use of the Maintain program for managing them. For an introduction, read section 3.7 and appendix B.2 of the *Laurel Manual*, filed as <Laurel>Laurel.press on most file servers. Complete instructions for using the Maintain program are included in the *Maintain Reference Guide*, file <Laurel>Maintain.press.

3.1. User names and passwords

Ordinarily, you log into an IFS with the user name and password by which you are known to Grapevine. If you are a Laurel user, this is the same as the name and password you use with Laurel. If you change your Grapevine password using the Maintain program, IFS will recognize your new password automatically.

It is always correct to use your full registered name (R-Name), which is in the form *simpleName.registry*—for example, 'Jones.PA' or 'Smith.ES'. Additionally, each IFS has a default registry which is the Grapevine registry of most of its local users; for example, the default registry for the Ivy and Indigo file servers is 'PA'. Users whose R-Names are in the default registry need not specify the registry name.

You probably have your own directory on at least one local IFS, for storage of personal files. This directory's name is generally the same as the *simpleName* portion of your R-Name. Once you have logged in, you are given control of the directory whose name either matches your full R-Name or can be extended to match your full name by appending a period and the IFS's default registry name. For example, on an IFS whose default registry is PA, user Jones.PA gets control of the directory whose name is 'Jones', but user Jones.ES does not; however, on the same file server there can also be a directory named 'Jones.ES' which user Jones.ES controls and Jones.PA does not.

You probably don't have your own directory on any IFSs besides the ones belonging to your local organization. However, you may still log in to any IFS that is using Grapevine for authentication; in such a situation, you won't have the right to store personal files on that IFS, but you may access files in other directories if their file protections permit.

You will probably find it convenient to install your full R-Name on your Alto disk (or in your Pilot User.cm, or whatever) instead of just your *simpleName*. This enables you to log into remote IFSs with the same user name you use locally.

3.2. User groups

Your access to files and directories is permitted or denied on the basis of your membership in *user groups*. Most authentic users are members of a special user group called 'World' (discussed in more detail below). 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 whose names and members are registered with Grapevine. Such user groups generally correspond either to organizations or to specific projects. A group's name is assigned by an administrator responsible for the registry; the name is an R-Name whose *simpleName* portion typically ends in '↑', as in CSL↑.PA or IFSAdministrators↑.PA. A group's membership is controlled by the group's owners; ownership of a group can be as restrictive or as open as the administrator deems appropriate.

You are probably already familiar with Grapevine groups: many of them are used as distribution lists for Laurel messages. For example, the group CSL↑.PA is a list of members of the PARC Computer Science Laboratory. If CSL↑.PA is specified as a recipient of a message, then all members of the group receive a copy. Similarly, if an IFS file's protection specifies that the file is

readable by CSL↑.PA, then all members of the group are permitted to read that file. The same group may be used for both purposes.

There is a fixed set of groups that may be used in specifying IFS file protections; this set is determined on a system-wide basis by the IFS's administrator, and there are at most 62 such groups. All the acceptable group names are displayed by the Show System-Parameters command, and are also displayed as alternatives if you type '?' when IFS is expecting you to type a group name.

You may examine groups used for IFS access control just the same as groups used as distribution lists, namely by running the Maintain program in Laurel. Similarly, you may request to be added to a group by contacting the group's owner; one way to do this is to send a message to 'Owners-*x*', where *x* is the full name of the group (e.g., 'Owners-CSL↑.PA').

There is not actually a Grapevine group called 'World'. Rather, 'World' is a shorthand name for a special group; for most IFSs in the United States, this group is USRegistries↑.internet, which includes all registered Xerox employees in the U.S. and Canada. Foreign individuals are excluded from 'World' in order to satisfy U.S. technology export regulations; foreign access to U.S. information is expected to be conducted through more formal and controlled channels. The definition of 'World' is controlled on a system-wide basis by the IFS's administrator; it is displayed by the Show System-Parameters command.

3.3. File protections

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. Append access is not presently implemented.

The standard default file protection permits read, write, and append access to the Owner and read access to the World (i.e., to members of the IFS's 'World' group, described above). 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 (assuming you are a member of the 'World' group). But, for example, if the file protection also permits write access by group CSL↑.PA, and you are a member of group CSL↑.PA, 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. (Once connected to a directory, you have the same privileges as that directory's owner.) As with file protections, these types of access are granted or denied individually to Owner, World, and each 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.

3.4. Commands dealing with protections

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 (which are a subset of the sub-commands of the Change Attributes command):

@@ 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 names (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: CSL↑.PA Owner; 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. This command also displays some information about user group membership; this reflects information that IFS has determined by querying Grapevine, and is not necessarily accurate or up-to-date.

3.5. Non-Grapevine users and groups

In an IFS that does not use Grapevine for authentication and access control, the IFS must maintain local information about user names, passwords, and group memberships. Additionally, even in an IFS that does use Grapevine, there may be some non-Grapevine user names and groups. (In particular, the names and passwords of files-only directories are typically not registered with Grapevine but are maintained solely by the IFS.) The commands described in this section are used to manipulate the local authentication and access control data base.

Note that IFS uses the same data structures to keep temporary copies of Grapevine information as it uses to keep permanent local information. That is, the commands described below can be used to manipulate Grapevine as well as non-Grapevine user names and group memberships. However, using these commands to examine Grapevine R-Names will in general yield information that is incomplete or out-of-date; and attempts to make changes to the local data base will not have permanent effects. You must use Maintain to get correct information or make permanent changes involving Grapevine R-Names.

The password of a non-Grapevine directory may be changed by the following command:

@ 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 is displayed as you type it; this is so that if you make a typing mistake you will be able to see it.)

An IFS administrator can assign you to be the *owner* of one or more non-Grapevine 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 execute, because it has to read the directory parameters of every user in the system.

In addition to having names, groups have numbers in the range 0 to 61. The association between group numbers and names is displayed by the Show System-Parameters command. A group can be used for protection purposes even if it doesn't have a name; all commands that accept group names also accept group numbers.

4. Other servers

4.1. Mail server

IFS optionally makes available a mail server compatible with the Laurel message system interface and the Grapevine transport mechanism. Most registries are now served by Grapevine servers. However, there are a few registries that still use an IFS as its sole mail server; also, an IFS can serve as an adjunct to a Grapevine registry by keeping some of its mailboxes.

It is largely invisible to a Laurel user whether one's mailbox is kept by Grapevine or by an IFS.

4.2. 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.bfs'.

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 '.bfs', which stands for Basic File System.

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.

4.3. Leaf server

IFS optionally makes available a server for the 'Leaf' page-level access protocol, which permits random access to parts of IFS files as opposed to the transfer of entire files. There are several user programs that take advantage of this capability, though these programs are still experimental and not widely available.

Not all IFSs run Leaf servers. An IFS that is running Leaf identifies itself by an 'L' suffix on the IFS version number in the herald displayed at connection time—for example, '1.36L'.

5. File backup

Reliability of file storage is accomplished by two facilities. 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.)

6. 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 (recent history notwithstanding). 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. Steve Butterfield initially implemented the Mail and Leaf facilities and made some important internal improvements. Ted Wobber contributed improvements to the Press printing facility and presently maintains the Leaf server.

Inter-Office Memorandum

To	IFS Administrators	Date	October 3, 1982
From	Ed Taft	Location	Palo Alto
Subject	IFS Operation (version 1.37)	Organization	PARC/CSL

XEROX

Filed on: <IFS>Operation1 & 2.bravo, <IFS>Operation.press

This memo describes operating procedures for the Interim File System software. It assumes that you are familiar with standard Alto software in general and with IFS from the user's point of view (<IFS>HowToUse.bravo).

A short summary of revisions to this document may be found at the end. Also, section 14 contains a summary of known bugs in the current release of the software and how to avoid them.

The current release of IFS should be run under OS version 20 or later. For correct error reporting, you must have current versions of Swat and Sys.errors.

1. Hardware requirements and organization

IFS requires a standard Alto with a Trident disk controller and one to eight Trident T-80 or T-300 disk drives in any combination. A T-80 disk pack holds 36,675 pages of 1024 words each, while a T-300 holds 139,365 pages. Due to a software limitation, only 130,986 pages of a T-300 are accessible.

The software deals with a *primary* file system consisting of one or more disk packs. A multiple-pack IFS behaves logically as a single file system, and all packs must be present and on-line in order to access any files. Files created by IFS are not accessible to other Trident-based programs (e.g., TFU, FTP) or vice versa.

Additionally, the software can deal with one or more *secondary* file systems that may be mounted and dismounted while IFS is running. An example of a secondary file system is a disk pack used for on-line, incremental backup.

The number of disk drives needed to support a given size file system depends both on how backup is to be accomplished and what degree of redundancy is desired in case of disk drive failure. Backup procedures are discussed in a later section.

Any Alto (Alto-I or Alto-II) with a Trident disk controller will run the IFS software; however, for performance reasons, it is strongly recommended that an Alto-II with at least 128K words of memory be used. See section 13.2 for details. Beginning with of IFS 1.36, we have abandoned the practice of packaging the code to minimize working set in a 64K configuration. Certain common operations such as FTP Store now have a working set greater than 64K, so a server with only 64K of memory is likely to thrash very badly even under light load.

There is an Alto-II hardware modification that is recommended for machines that are to be IFSs (or other servers requiring high reliability). This modification corrects a design bug in the memory single-error correction. The memory chips are so reliable that the single-error correction is hardly ever invoked; but servers that run for months at a time eventually exercise even low-probability bugs. The modification is described in [Maxc2]<IFS>MemECMod.press.

the dialogue.

IFS now initializes the file system, an operation that takes about 2 minutes per T-80 and 7 minutes per T-300 in the system. When the screen turns black and the cursor changes from an hourglass to 'IFS', initialization is complete.

The file system initially has only three directories defined: System, Default-User, and Mail. The password for System is IFS. You should connect to the IFS using Chat, login as System (password IFS), create some other users (by means of the Create command, described below), and change System's password for the sake of security.

Before putting the system into service, there are various system parameters you must set; these are described in later sections. They include:

clock correction and server limit (section 7);

switches to enable or disable Press printing and the Boot, Name, Time, Leaf, CopyDisk, and LookupFile servers (sections 7 and 9);

switches, default registry, and group names for Grapevine authentication and access control (section 6);

mail system parameters (section 8);

backup system parameters (section 11).

5. Normal operation

The system is normally started simply by invoking IFS with no switches. All file system packs must be mounted and on-line, and *all Read-only switches must be turned off* (including those on backup and spare drives). It is unimportant which packs are mounted on which drives, since the software reads each pack to discover what the system configuration is. (However, there must be no other primary IFS packs on-line. After copying an IFS pack with CopyDisk, you should be careful to remove the copy from the system.)

If IFS fails to start up properly, it will call Swat with an appropriate error message. This will occur, for example, if all necessary disk drives are not on-line. While IFS is starting up, the cursor contains an hourglass; this changes to 'IFS' when startup is complete and the system is in operation.

During startup, IFS also runs a brief test of the Control RAM and S-registers, failure of which will cause IFS not to function even though a great deal of other Alto software works correctly. IFS will call Swat with an appropriate message if this test fails.

When IFS is started, it verifies the consistency of the directory B-Tree (unless inhibited by '/-V'). Inconsistencies can result from crashes at inopportune moments when the B-Tree is in the process of being modified, so the startup-time check is valuable in determining whether it is appropriate to run the IFS Scavenger. The time required for the check is proportional to the number of files in the system; it has been observed to take 2 minutes for 20,000 files.

If an inconsistency is detected, IFS will call Swat with an appropriate message. The message 'Record count disagrees' is relatively benign and it is reasonably safe to proceed from this (with control-P). If you do so, it is likely that one or more files in the file system will become inaccessible and will remain so until the next time the IFS Scavenger is run. Other possible errors include 'Records out of order' and 'Malformed B-Tree record'; these are more serious and proceeding is *not* recommended.

Immediately after IFS starts, it will perform some other initialization operations requiring a lot of disk activity, including obtaining and installing the network directory (name server data base) and boot files. IFS can service user requests during this time (5 minutes or more), but performance will be noticeably poorer than normal.

While IFS is running, the entire screen is black except for the cursor. The position of the cursor is an indication of disk activity. The horizontal position indicates the disk unit most recently accessed (unit zero at the extreme left, unit seven at the right), and the vertical position is the cylinder at which the heads are currently positioned (zero at the top, 814 at the bottom). The cursor blinks each time a page is transferred to or from a user file by the file server.

You can tell whether IFS is running normally by pressing the space bar. If the Alto screen flashes, everything is in order; if not, the system has failed in some way.

There are two ways to stop the system. The normal (and cleaner) way is to connect to IFS using Chat, log in, Enable, and issue the Halt command. The system will then refuse to admit further users, will wait for all present users (including you) to log out or disconnect, and will return control to the Alto Executive.

The system may also be stopped by typing Shift-Swat on the IFS Alto keyboard (the Swat key must be pressed firmly). This aborts all active connections and returns control to the Executive immediately; however, it may leave partially-written files lying around so it is not recommended for normal use.

6. Directory and user group management

This section assumes that you thoroughly understand IFS's use of Grapevine for authentication and access control, described in the 'How to use IFS' document. Additionally, before beginning to create IFS directories and groups, you should read 'Access Controls', file [Maxc]<IFS>AccessControls.press, for a description of the policies and procedures required to ensure proper information security.

6.1. Setting up user directories

The IFS Executive (accessed via Chat) has several privileged commands that are available only to users with the 'wheel' capability, and only after enabling this capability with the Enable command. When you are so enabled, the Executive's prompt is '!' rather than '@'. While you are enabled, IFS will not log you out automatically after three minutes of inactivity as it does normally.

The following privileged commands are defined.

! Create (directory) *directory-name* [new] (password) *password*
!! *sub-commands*

> Creates a directory with the supplied name and password. Capitalization of the name should be precisely as the user wants to see it. Capitalization of the password is unimportant.
>
> If Grapevine authentication is enabled, *directory-name* may or may not be a fully-qualified R-Name of the form *simpleName.registry*. For a local user whose registry is the same as the IFS's default registry, it is best to omit the *.registry* when creating the directory, and just use the *simpleName*. However, for a user whose registry is different from the IFS's default registry, the directory name must consist of the full R-Name, else Grapevine authentication won't work properly.
>
> In general, names for files-only directories should *not* be qualified by registry (and shouldn't be registered in Grapevine either).
>
> For a directory that corresponds to a Grapevine R-Name, the *password* you supply is irrelevant, since IFS consults Grapevine for authentication. For a non-Grapevine name, if you specify an empty password, then the password cannot be used to gain access to the directory. This is useless for a login directory; but it is useful for a files-only directory to which access is granted solely on the basis of group membership rather than on users

knowing the directory's password.

The *sub-commands* are used to change various parameters from their default values. These include all the sub-commands of the Change Directory-Parameters command, described in 'How to use IFS', and additionally include:

!! Files-only (owner) *user-name*

Declares the directory to be a files-only (i.e., non-login) directory. The *user-name* is the person responsible for this directory; that user is permitted to connect to the directory without giving a password. *User-name* must include a *.registry* part if the user's registry is different from the IFS's default registry, but may be omitted if it is the same as the IFS's default registry.

!! Disk-limit *number*

Specifies the maximum number of disk pages that may be used in this directory.

!! Wheel

Declares the user to have the 'wheel' capability, which permits issuing privileged commands and bypasses all access checking and disk limits.

!! Mail

Creates a mailbox, thereby enabling IFS to receive Laurel mail for this user. (More information about mail is presented in section 8.)

!! Group Membership (in groups) *groups*
!! Group Ownership (of groups) *groups*
!! No Group Membership (in groups) *groups*
!! No Group Ownership (of groups) *groups*

Establishes the user's membership in or ownership of user groups. (The Group Membership sub-command duplicates the function of the top-level Change Group-Membership command.) Use of these commands is meaningful only for non-Grapevine members of non-Grapevine groups.

Note that in an IFS that does not use Grapevine for access control, any user able to log into the IFS is automatically a member of World, regardless of the setting of the directory's group membership. In an IFS that does use Grapevine for access control, a user whose names is registered in Grapevine is a member of World or not according to whether he is a member of the IFS's 'World' group, usually USRegistries↑.Internet. A user whose name is *not* registered in Grapevine but who *is* able to log into the IFS (because an IFS directory by that name exists) is a member of World only if so specified by use of the Group Membership command.

If *directory-name* already exists, you are not asked for a password but rather are sent directly into subcommand mode. This permits modifying parameters for an existing directory. In this context, the following additional subcommands are of interest:

!! Password *password*

!! Not Files-only

!! Not Wheel

!! Not Mail

Note that to make a non-Grapevine directory unusable for login purposes, you should issue the Password sub-command with an empty *password.* The top-level Change Password command cannot be used for this purpose.

The Create command is terminated by typing two Returns in response to the '!!' subcommand prompt. You may cancel the entire command by typing control-C.

The default values of all parameters for new directories are copied from a dummy directory called Default-User. When a file system is created, the default values are Not Files-only, Not Wheel, Not Mail, and Disk-limit 1000. To change the defaults, use the Create command to modify the parameters for the directory Default-User. 'Not Mail' is the default for files-only directories, even if Default-User specifies 'Mail'.

! Destroy (directory) *directory-name* [Confirm]

Destroys the specified directory. This operation includes deleting all the files contained within it, and destroying the associated mailbox if there is one.

! Change Directory-Parameters (of directory) *directory-name*
! Show Directory-Parameters (of directory) *directory-name*

These commands work as described in the 'How to Use' document with the addition that while you are enabled, you may access any directory, not just your own. Also, while you are enabled, Change Directory-Parameters has all the sub-commands described above under Create.

6.2. Setting up Grapevine authentication and groups

The commands in this section are used to control IFS's use of Grapevine for authentication and access control.

! Change System-Parameters
!! *sub-commands*

Permits you to issue sub-commands to change one or more system operating parameters. Each sub-command takes effect immediately. Sub-command mode is terminated when you type CR in response to the '!!' prompt.

!! Default-Registry *registry*

Sets the default Grapevine registry to be *registry*—e.g., PA, ES, etc. This should be the registry to which the majority of the local users of the IFS belong. Names of directories belonging to these users need not be (and generally should not be) qualified by the registry name; but names of directories belonging to users in other registries must be qualified by registry name.

You should set a default registry even if you are not using Grapevine for authentication and access control. This is to permit local users to present their user names either with or without registry qualification.

Note: if the mail server is enabled (section 8), *registry* must be the same as the mail server registry specified by the Registry sub-command of the Mail command.

!! Enable Grapevine Authentication !! Disable Grapevine Authentication

Enables and disables IFS's use of Grapevine for authentication. When this is enabled, any user whose name is registered in Grapevine and who is able to supply the correct (Grapevine) password can log into IFS. When this is disabled, only users who have personal directories on the IFS can log in, and they must present the password maintained by that IFS.

Before enabling Grapevine authentication, you must specify a default registry as described above.

!! Enable Grapevine Groups !! Disable Grapevine Groups

Enables and disables IFS's use of Grapevine for access control (group membership checking). When this is enabled, users' access to files is controlled by membership in Grapevine groups, as discussed in 'How to use IFS'. When this is disabled, user group membership is maintained entirely within the IFS, and is controlled by the Change Group-Membership or Change Directory-Parameters commands.

Before enabling Grapevine group checking, you must specify a default registry as described above, and you must associate Grapevine group names with IFS group numbers as described below. It is probably not sensible to enable Grapevine group checking without also enabling Grapevine authentication.

!! Group (name of group) *group-number* (is) *group-name*

Associates the Grapevine *group-name* with the IFS *group-number*, which is a number in the range 0 to 61. The *group-name* must be a fully-qualified name acceptable to Grapevine as either a real group (e.g., CSL↑.PA) or a pseudo-group (e.g., *.PA).

Caution: if you use this command to associate a name with a group number that already has a name, the meanings of all existing protections that refer to that group will be changed. For example, suppose group 3 is originally associated with the name CSL↑.PA, and some files' protections are set to permit reading by members of group CSL↑.PA. If you then change group 3 to associate it with the name PARC↑.PA, those files will become readable by all members of PARC↑.PA.

It is permissible for a group to have a name that is not registered in Grapevine; however, such a group can only contain members whose names are also not registered in Grapevine, and their membership must be managed by use of the IFS Change Group-Membership and Change Directory-Parameters commands. Consequently, this mode of use is really suitable only for IFSs in which Grapevine group checking is disabled.

To eliminate a group's name (i.e., make the group number no longer have a name), use the Group command as described above, but specify an empty *group-name*. That is, when IFS prompts you for the *group-name* and suggests the existing one as a default, erase the existing name using CTRL-W and then strike CR.

!! World (group is) *group-name*

Defines the membership of the 'World' group for this IFS. An appropriate value of *group-name* for most IFSs at Xerox sites in the U.S. is 'USRegistries↑.Internet'. If this is unspecified or empty, any user able to log into the IFS is considered a member of 'World'.

Note: this restricted definition of 'World' is enforced only if Grapevine

group membership checking is enabled.

6.3. Converting an existing IFS to use Grapevine

This section outlines the procedure for converting an existing IFS to use Grapevine for authentication and access control. The general idea is to extract the existing authentication information and group structure from the IFS and then capture this state in the Grapevine data base. Some of the information here is also relevant when starting up a new IFS.

6.3.1. Use the Accountant program to obtain an accounts listing and a group membership summary, as described in section 12.

6.3.2. Decide what the default registry is to be. This will usually be obvious: the Grapevine registry of the majority of the local users of the IFS. Set this by means of the Default-Registry sub-command of Change System-Parameters.

6.3.3. For each login (non-files-only) directory in the IFS, make sure that '*directoryName.defaultRegistry*' is registered as an individual in Grapevine, and that the Grapevine entry indeed represents the same user as the IFS directory's owner. For most local users this will already be the case, assuming that Grapevine is in regular use locally for mail service. (Note: you need not take any action on the built-in directory names Default-User, Mail, and System.)

In the case of a login directory belonging to a user in another registry, you may take one of three actions:

1. Most commonly, such a directory exists solely to give that user access to files in other directories on the same IFS, and the user does not store any files at all on his own directory. In this case, simply destroy the directory and instruct the user to use his own full R-Name when logging in. If the user is a member of any IFS user groups, you should make sure that the user's full R-Name appears in the corresponding Grapevine groups (below).

2. If the remote user actually uses his own directory for storage of files on the IFS, you should change the directory name to include the registry qualification. For example, if the IFS's default registry is PA, and user Jones.ES has a directory whose name is simply 'Jones' (or 'JohnJones' or something), then the directory's name should be changed to 'Jones.ES'. Unfortunately, the only way to do this is to create a new directory, move the files from the old directory to the new (using FTP or Brownie), and destroy the old directory.

3. This alternative is not recommended, but is sometimes workable in desperate situations. If the result of appending the IFS's default registry to the directory name yields an R-Name that is *not* registered in Grapevine, then you can leave the directory as-is. The disadvantages of this alternative are that the user cannot gain access to files by virtue of his R-Name being in Grapevine groups, and the directory name may conflict with a name registered in Grapevine in the future. Note: you may need to explicitly specifiy the directory's membership in the World group, as discussed above in the description of the Group Membership sub-command of Create.

Since all registered users are able to log in under their full R-Names, 'Guest' accounts are no longer needed and should be abolished. This will result in a substantial improvement in information security. Accounts that permit automatic access by programs using compiled-in credentials should likewise be abolished; procedures for dealing with such situations are described in the 'Access Controls' memo.

6.3.4. In general, the names of files-only directories should *not* be registered in Grapevine. Ordinarily, users gain access to a files-only directory by virtue of membership in user groups referenced by that directory's connect protection, not by presenting the directory's password. However, so long as the result of appending the IFS's default registry to the directory name yields an R-Name that is *not* registered as an individual in Grapevine, the directory may be left as it is; IFS will use local information to authenticate users' attempts to connect to it. (If you are so

unfortunate as to encounter a files-only directory whose name *is* registered in Grapevine, then either users must use the Grapevine password to connect or you must change the name of the directory.)

6.3.5. Issue the Enable Grapevine Authentication sub-command of Change System-Parameters. IFS will now use Grapevine for user authentication, while continuing to use local user group information for access control.

6.3.6. You will need the cooperation of the Grapevine registry's maintainer to perform this step. Compare the IFS group membership information obtained in step 6.3.2 with the existing Grapevine groups, using the Maintain program. In most cases you will find that some existing Grapevine group is substantially the same as the IFS group, and that any discrepancies are most likely mistakes or oversights. (This is particularly true of organization and project groups.) In such cases, adjust the Grapevine group membership as necessary and use it for the IFS group. For the remaining IFS groups, create new Grapevine groups with the same memberships as the IFS groups. Remember that each name in a Grapevine group must include a registry name, *even when the registry name is the same as the IFS's default registry.*

The groups used for IFS access control should be chosen with some care. The simplest groups that will do the job should be used. Grapevine takes a long time to check for membership in extremely large or complex groups (e.g., SDD↑.ES, which is both large and complex); this has the potential for causing severe performance bottlenecks. In this connection, it should be mentioned that pseudo-groups of the form '**.registry*' (e.g., *.PA, *.ES) can be checked extremely rapidly, in contrast with groups such as AllPA↑.PA which require lengthy enumerations.

It is OK for a group to refer to other groups, but the *simpleNames* of those other groups must end in '↑', else they won't be expanded by Grapevine when checking for group membership. (However, it is not required that the top-level group name contain '↑'.)

Now use the Group sub-command of Change System-Parameters to associate the chosen Grapevine group names with the in-use IFS group numbers. Again, each group name must be a full R-Name, including registry.

You should do this step carefully to ensure that you have captured in Grapevine the existing IFS group structure, because step 6.3.8 will invalidate and destroy the IFS group structure.

6.3.7. Decide on a definition of the 'World' group. This controls access to all files (both existing and new) whose protections specify 'World'—which is to say, all files accessible to Guest in IFSs that have had Guest accounts until now.

'World' is intended to be an all-encompassing group that permits limited access to public files by all authentic users. The main reason for restricting 'World' membership is to satisfy U.S. technology export regulations. Several non-U.S. Grapevine registries are now being established, and foreign access to U.S. information is expected to be conducted through more formal and controlled channels. (This topic is discussed in detail in the 'Access Controls' memo.)

Therefore, the definition of 'World' appropriate for most IFSs in the U.S. is a special group called USRegistries↑.Internet, which includes all members of all U.S. registries. Certain IFSs may choose to adopt a more restricted group for 'World'. Note that a user who is not a member of an IFS's 'World' group can still be a member of other of that IFS's groups and can access files whose protections mention those groups.

Use the World sub-command of Change System-Parameters to set the name of the IFS's 'World' group.

6.3.8. Issue the Enable Grapevine Groups sub-command of Change System-Parameters. The conversion to Grapevine is complete.

6.3.9. You may now delete all user directories that have disk limits of zero. Such directories presumably exist solely to give the users access to other directories on the IFS; since authentication and access control are now done using Grapevine, these directories are no longer needed. *Exception:* directories that have special capabilities ('Mail' or 'Wheel') should *not* be deleted, as

these capabilities are remembered only by the IFS and not by Grapevine.

It is worth mentioning the measures IFS takes to prevent Grapevine from becoming a bottleneck and to enable continued service if all Grapevine servers become unavailable. When a user first logs in or first attempts to access a file in a way that requires membership in some group, IFS interacts with Grapevine in whatever ways are necessary and remembers the result. This means that subsequent logins or file accesses involving the same group membership do not require any interaction with Grapevine.

The Grapevine information that IFS remembers locally is invalidated after 12 hours; this prevents IFS from getting more than 12 hours out of date with respect to Grapevine. Furthermore, IFS remembers only *successful* user accesses, not unsuccessful ones. This means that when a new Grapevine R-Name is created for a user, or a user is added to a Grapevine group used for access control by IFS, the changes have an immediate effect on the user's access to IFS. However, when an R-Name or group member is removed, IFS may continue to allow access on the basis of the obsolete information for up to 12 hours.

Finally, if Grapevine becomes inaccessible for an extended period, IFS will continue to use its remembered Grapevine information indefinitely.

7. Other privileged commands

! Disable

Leaves enabled mode (the Executive's prompt reverts to '@').

! Halt

Stops IFS and returns control to the Alto Executive as soon as all present users (including you) log out or disconnect. If any Leaf connections are active, there may be a delay of several minutes before the connections are broken and IFS halts.

! Show Printing-requests (for user) *user*
! Cancel (printing requests) (for user) *user*

If you type just RETURN in place of *user*, these commands will run through all printing requests for all users.

! Change System-Parameters
!! *sub-commands*

Permits you to issue sub-commands to change one or more system operating parameters. Each sub-command takes effect immediately. Sub-command mode is terminated when you type CR in response to the '!!' prompt.

The following sub-commands have permanent effects that survive restarts of IFS. (The permanent information is kept in file <System>Info!1 in the IFS's primary file system.)

!! Clock-Correction *correction*

Sets the software clock correction, which is specified as a sign (+ or −) followed by a decimal number. This causes the Alto clock to run faster (+) or slower (−) than its nominal rate by that number of seconds per day. Alto clocks are quite stable but not particularly accurate, and software correction is desirable in a server that runs continuously for long periods of time.

The amount by which the clock should be corrected may be determined by comparison with an accurate reference over a period of several days (using

the IFS Executive's 'DayTime' command), or by use of an accurate frequency counter to measure the Alto system clock (slot 5 pin 63 on an Alto-I, slot 13 pin 63 on an Alto-II) and computing the correction by the formula

$$c = 86400 * (1 - f / 5880000)$$

where f is the frequency in Hz.

!! Server-Limit *n*

Limits the number of simultaneous server processes (FTP, Chat, and Mail combined) to *n*. At present, *n* may be as high as 6 on an Alto with 64K of memory, 8 with 128K, and 10 with 192K or more; the default Server-Limit is one less than the absolute maximum. See section 13.2 for information on setting this parameter properly.

!! Enable Press-printing
!! Enable CopyDisk-server
!! Enable Boot-server
!! Enable Name-server
!! Enable Time-server
!! Enable LookupFile-server

!! Disable Press-printing
!! Disable CopyDisk-server
!! Disable Boot-server
!! Disable Name-server
!! Disable Time-server
!! Disable LookupFile-server

Enables and disables the Press printing facility and various servers (the FTP server is always enabled, and the mail system is controlled by subcommands to the Mail command). When a file system is created, all the servers are disabled.

!! Enable New-boot-files

!! Disable New-boot-files

This controls the operation of the automatic boot file maintenance mechanisms, and is relevant only if the boot server is enabled. If New-boot-files is enabled, IFS will obtain and maintain copies of all boot files available from any other boot servers on the same Ethernet. If New-boot-files is disabled, IFS will maintain only those boot files that it already has. When a file system is created, New-boot-files is enabled.

!! Enable Leaf-server

!! Disable Leaf-server

Enables and disables the Leaf page-level access server (see section 9.4). The Enable command does not take effect until IFS is next restarted, unless IFS was last restarted with the /L switch. Similarly, the Disable command does not free resources consumed by the Leaf server until IFS is next restarted. When a file system is created, the Leaf server is disabled.

The following sub-commands have one-time-only effects.

!! Disable Logins

Disallows further user access to the system. This is useful during debugging and while reloading the file system from backup.

!! Enable Logins

Cancels the effect of Disable Logins.

!! Reset-Time

Causes IFS to reset its clock from a time server on the directly-connected

Ethernet. This operation is performed once automatically, immediately after IFS restarts.

8. Mail system

IFS contains a mail server that is compatible with Laurel and the Grapevine message system. IFS can keep in-boxes for users of that system and can also forward mail to mail servers in other Grapevine servers, IFSs, and Maxc.

If your IFS is in the Xerox Research Internet and you wish to operate a mail server, you should first consult the network support organization <NetSupport.WBST> to obtain useful advice. A mail server must be assigned a *registry name* that is distinct from the name of the IFS Alto itself. An IFS can be either a self-contained registry or an adjunct to a Grapevine registry.

To enable a user to receive mail, issue the Mail subcommand of the Create command, as described previously. Of course, if you turn on the Mail capability for Default-User, then all new user accounts you create subsequently will have Mail capability automatically.

When mail is received from a user mail program such as Laurel, it is queued briefly in files named '<Mail>New>Mail!*'. A process called MailJob then wakes up and distributes the messages to individual in-boxes, which are files named '<Mail>Box>*user-name*!1'. When a user retrieves the mail from his in-box, the in-box file is reset to empty. If you disable the mail capability for a user who has mail pending in his in-box, the in-box file will be deleted next time it is read.

The MailJob process also forwards mail to other mail server hosts. Specifically, messages addressed to *user.registry*, where *registry* is the name of some other mail server, will get forwarded to that server by the Mailer process. While being forwarded, such messages are queued in files named '<Mail>Fwd>*registry*'. If *registry* maps to multiple addresses, as does 'GV' and other Grapevine registries, then the Mailer will try each of those addresses, closest first, until it succeeds in delivering the messages.

When a file system is first created, the mail system is disabled. You should set the enable switches and configure the mail system using the commands below.

8.1. Mail system commands

The mail system is controlled by a command processor which is entered via the privileged Mail command to the IFS Executive. The commands that change parameters take effect immediately and survive restarts of IFS.

* Enable (mail) System
* Disable (mail) System
* Enable (mail) Forwarding
* Disable (mail) Forwarding

Enables and disables the mail system. The first command turns on and off the mail system as a whole; the second command enables or disables forwarding of mail to other mail servers.

* Dead-letter (recipient name) *name*

Specifies the name of the in-box to which notification of mail system problems (e.g., undeliverable messages with no return address) should be directed. *Name* may be the name of an in-box on this IFS or (if forwarding is enabled) the fully-qualified name ('*user.registry*') of a recipient on some other server. If the IFS has access to a Grapevine server, 'DeadLetter.MS' is a good value for *name*.

* Distribution-lists (directory name) *name*

Specifies the name of the directory that holds distribution lists (extension '.dl') that are to be remotely accessible via the mail server. *Name* must *not* be enclosed by '<'

and '>'; that is, if you keep distribution lists as '<System>DLs>*.dl', you should specify 'Distribution-lists System>DLs'.

This mechanism is used to keep distribution lists only for non-Grapevine registries. For Grapevine registries, distribution lists (groups) are maintained exclusively by Grapevine, even if IFS keeps some of the mailboxes for that registry.

If *name* is empty, the distribution list retrieval mechanism is disabled.

* Grapevine (server name) *name*

Specifies the name of a Grapevine server (ordinarily 'GV'). Mail to a user at a remote registry is forwarded to *name*. Mail to a user whose registry maps to this IFS or matches the registry system parameter, but for which no local mailbox exists, is forwarded to *name*. If *name* is empty, mail for a remote user is forwarded to its registry, and in the second case is returned as undeliverable.

* Registry (we are part of) *name*

Specifies the name of a Grapevine registry to which this IFS belongs. Mail to a user whose registry matches *name* is treated as if the recipient's mailbox is local. If no local mailbox exists and a Grapevine name has been specified, then the mail is forwarded there. If *name* is empty then this IFS's registry name is determined by looking up its machine address (mail server socket 7). Note: if Grapevine authentication is enabled, *name* must be the same as the default registry established by the Default-Registry sub-command of Change System-Parameters.

* Status

Gives the current status of the mail system, and a summary of mail server operating statistics covering the interval since the file system was created or last reloaded. Statistics are kept on five items; for each item three things are recorded: 'Samples', the total number of times a statistic for that item was recorded; 'Avg', the average value of the statistic; and 'Histogram', an eight slot, logarithmically scaled histogram of the values.

Length (characters)

This is the length of a message received by the mail server, including header text. A message with multiple recipients is counted once in this statistic.

Recipients

This is the number of recipients ('To:' plus 'cc:') of a message received by the mail server.

Sort delay (sec)

This is the time between receiving a message and appending copies to local in-boxes or queueing copies for forwarding. It is recorded once for each local recipient and once for each remote mail site.

Fwd delay (sec)

This is the time between queueing a message for a remote mail server and successfully delivering it to that server. The statistic is recorded once for each message forwarded, even if the message is addressed to multiple recipients at the given remote server.

Retrieve delay (min)

This is the time between the mail system's appending a copy of a message to a local in-box and the owner's retrieving it.

If any messages have been discarded by the mail system, the number of occurrences will be displayed. A message is discarded when it can't be delivered to a recipient and it can't be returned to its sender and it can't be delivered to the dead letter destination. This is an indication of serious trouble.

* Reset (mail statistics)

Resets the mail statistics, which are kept continuously and ordinarily survive restarts of IFS.

9. Other servers

IFS contains various servers that provide essential services to other hosts on the directly-connected Ethernet. These include the 'miscellaneous' servers (principally boot, name, and time), the Leaf page-level access server, the CopyDisk server, and the LookupFile server.

The boot, name, and time server functions duplicate those provided by gateway systems, so in a network with at least one gateway, it is not necessary for IFS to provide these services. But in a network that includes an IFS and no gateways, it is necessary for the IFS to provide the services. Even in networks that do have one or more gateways, running the IFS miscellaneous servers may be advantageous in that the availability of the services is improved. (Also, it should be noted that the IFS boot server is noticeably faster than the boot servers of existing gateway systems.)

Since running the miscellaneous servers may slightly degrade the performance of an IFS in its principal functions, means are provided to turn them off (the Change System-Parameters command, described in section 7). When a file system is first created, all the miscellaneous servers are disabled.

IFS participates in the protocols for automatic distribution and maintenance of the date and time, the network directory, and the common boot files. When IFS is started up for the first time, and thereafter whenever any changes are distributed, IFS obtains all necessary files from neighboring servers (gateways or other IFSs). The name server data base is maintained even if the IFS name server is disabled, because IFS requires it for its own internal purposes (principally mail forwarding).

9.1. Name server

The name server data base is kept as file '<System>Pup-network.directory'; a new version is created and older versions deleted whenever a new file is distributed. If there are no other name servers on the directly-connected Ethernet, you must use the BuildNetworkDirectory procedure to install new versions of the network directory.

9.2. Boot server

The boot files are kept in files '<System>Boot>*number-name*.boot', where *name* is the name of the boot file and *number* is its boot file number in octal (for example, '<System>Boot>4-CopyDisk.boot'). Standard boot files have centrally-assigned boot file numbers less than 100000 octal, and are distributed automatically. Non-standard boot files have boot file numbers greater than or equal to 100000 octal and are not distributed automatically.

Ordinarily, IFS will obtain and maintain its own copy of all standard boot files maintained by any other boot server on the same Ethernet. This is the appropriate mode of operation for most boot servers. However, in some situations it is desirable for an IFS boot server to maintain only a subset of all available boot files. The Disable New-boot-files sub-command of Change System-parameters may be used to enter this mode; subsequently, IFS will not obtain any *new* boot files, but will continue to maintain and update any boot files that it *already* has. Additionally, boot files with numbers in the range 40000 to 77777 octal will always be managed in this fashion, regardless of the setting of the New-boot-files switch. Also, a boot file will not participate in the update protocol if it has a different number than the correspondingly-named boot file in other boot servers. By this means, special versions of standard boot files may be maintained on particular servers without interfering with the update of the standard versions on all other servers.

You may install or delete boot files by manual means (e.g., FTP), keeping in mind the file name and boot file number conventions described above.

Additionally, the boot server supports the MicrocodeBoot protocol, used to boot-load microcode on Dolphins and Dorados. The microcode files use a different numbering scheme from normal boot files. For purposes of boot file update, microcode file number n is assigned boot file number $3000_B + n$.

The boot server is also capable of boot-loading Sun workstations. The Sun boot protocol identifies boot files by name rather than by number. However, Sun boot files must still be assigned numbers to control the boot file update process, as described previously. Users need *not* mention these numbers when invoking boot files.

The various boot protocols are documented in [Maxc]<Pup>EtherBoot.press.

9.3. Time server

You should not enable the time server unless you have first calibrated and corrected the Alto clock, using the procedure described in section 7.

9.4. Leaf server

IFS contains a server for the 'Leaf' page-level access protocol, which permits random access to parts of IFS files as opposed to the transfer of entire files. There are several user programs that take advantage of this capability, though these programs are still experimental and not widely available.

At present, the Leaf software is being made available on a 'use at your own risk' basis. While it is included in the standard IFS release, it is enabled only on some file servers. Leaf was developed by Steve Butterfield, and is presently maintained by Ted Wobber <Wobber.PA> rather than by Taft and Boggs, who are responsible for the remainder of the system. Inquiries and trouble reports concerning the use of the Leaf server should be directed to Ted.

For performance reasons, the Leaf server should not be enabled in an IFS that also supports heavy FTP, Mail, CopyDisk and Telnet traffic, and the IFS Alto should have at least 192K of memory.

The Leaf server is enabled and disabled by sub-commands to the Change System-parameters command (section 7). However, for the Leaf server, the IFS software has to do a substantial amount of memory management at startup time which is impractical to perform during normal operation. Therefore, the Enable/Disable Leaf-server commands do not take effect immediately but rather are deferred until the next restart of IFS. More precisely, the Enable Leaf-server command is deferred until the next restart unless Leaf was already enabled at the last restart or IFS was last restarted with the /L switch. The Disable Leaf-server command takes effect after a delay of at most 3 minutes, but memory resources consumed by the Leaf server are not reclaimed until the next restart of IFS.

9.5. CopyDisk server

IFS contains a server for the CopyDisk protocol, which permits one to copy disks to and from an IFS. A CopyDisk server is equivalent to an FTP, Mail or Telnet server when deciding whether to create an IFS job in response to a request-for-connection. The load placed on the system by a CopyDisk job is about the same as an FTP job, except that transfers between disk and net will typically last much longer (minutes rather than seconds).

The CopyDisk server is enabled and disabled by sub-commands to the Change System-parameters command (section 7).

9.6. LookupFile server

The LookupFile server provides a means of verifying the existence of a file and determining its creation date and length, using a protocol that is substantially less expensive than either FTP or Leaf. Unlike FTP or Leaf, this server provides information to *unauthenticated* clients. If available, this server is used heavily by the file cache validation machinery in Cedar, with performance considerably better (and imposing less load on the server) than the corresponding operation performed via FTP.

The LookupFile server is enabled and disabled by sub-commands to the Change System-parameters command (section 7). The LookupFile protocol is documented in [Maxc]<Pup>LookupFile.press.

10. Adding packs to the file system

The capacity of an existing file system may be increased by adding more packs to it. This may be accomplished by the following procedure.

Initialize and test a pack using 'TFU Certify' in the normal fashion (section 3). Then, with IFS running, mount that pack on any free drive and issue the command:

! Extend (file system) Primary (by adding drive) *d* [Confirm]

'Primary' is the name of the file system you are extending, and *d* is the drive on which the new pack is mounted. IFS now initializes this pack, an operation that takes about 2 minutes for a T-80 and 7 minutes for a T-300. When it completes, the new pack has become part of the file system.

Note that there is no corresponding procedure for removing a pack from a file system. To decrease the number of packs in a file system, it is necessary to dump it by means of the backup system, initialize a new file system, and reload all the files from backup. This procedure is also required to move the contents of a file system from T-80 to T-300 packs.

Note also that adding packs to a file system does not increase the amount of directory space available. The size of the directory is determined when you first create the file system; there is no straightforward way (short of dumping and reloading) to extend it. (More precisely, while the software will attempt to extend the directory automatically if it overflows, this will significantly degrade subsequent performance, and too many such extensions will eventually cause the system to fail entirely.) Therefore, it is important that you allocate a directory large enough for all expected future needs. Experience has shown that 1000 directory pages are required for every 25,000 files in the file system, but this is highly dependent on a number of parameters including average file name length.

11. Backup

There are three facilities available for assuring reliability of file storage and for recovering from various sorts of disasters.

The first facility is the IFSScavenger program. It is analogous to the standard Alto Scavenger subsystem. It reads every page in the file system, makes sure that every file is well-formed, and checks for consistency between files and directories. For safest operation, it should be run after every crash of the IFS program. However, since it takes a long time to run, in practice it should only be run when major file system troubles are suspected (in particular, when IFS falls into Swat complaining about disk or directory errors). The IFSScavenger is described in a separate memo, available as <IFS>IFSScavOp.Bravo (or .Press) on Maxc1.

The second facility is an on-line incremental backup system that is part of the IFS program itself. It operates by copying files incrementally to a backup file system (ordinarily a single disk pack) mounted on an extra drive. The file system is available to users while the backup operation is taking place (though backup should be scheduled during periods of light activity to avoid serious performance degradations). Use of the incremental backup system requires that there be an additional disk drive connected to the Alto, over and above the drives needed for the primary file system itself. The backup system is described in the next section.

The third facility is the CopyDisk program. To back up the file system, one must take IFS down and copy each of the file system packs onto backup packs. On a machine with multiple disk drives, one may copy from one drive to another, an operation that takes about 4 minutes per T-80 and 15 minutes per T-300 if the check pass is turned off. One may also copy disks over the Ethernet to

another Alto-Trident system, but this takes about five times as long.

At PARC we use the Scavenger and the Backup system; we no longer use CopyDisk for backing up IFS. Regular operation of either the Backup system or CopyDisk is *essential* for reliable file storage. We have observed several instances of Trident disk drive failures that result in widespread destruction of data. It is not possible to recover from such failures using only the IFSScavenger: the IFSScavenger repairs only the *structure* of a file system, not its contents.

11.1. Backup system operation

The backup system works in the following way. Periodically (e.g., every 24 hours), a process in IFS starts up, checks to make sure a backup pack is mounted, and sweeps through the primary file system. Whenever it encounters a file that either has never been backed up before or was last backed up more than *n* days ago (a reasonable *n* is 30), it copies the file to the backup pack and marks the file as having been backed up now. Human intervention is required only to change backup packs when they become full.

The result of this is that all files are backed up once within 24 hours of their creation, and thereafter every *n* days. Hence every file that presently exists in the primary file system is also present on a backup pack written within the past *n* days. This makes it possible to re-use backup packs on an *n*-day cycle.

Operation of the backup system has been made relatively automatic so as to permit it to run unattended during the early morning hours when the number of users is likely to be small. This is important because system performance is degraded seriously while the backup system is running.

11.2. Initializing backup packs

To operate the backup system, you need a disk drive and some number of packs dedicated to this purpose. The number of packs required depends on the size of your primary file system, the file turnover rate, and the backup cycle period *n*. The packs should have their headers and labels initialized using 'TFU Certify' in the normal fashion (section 3). Then they must each be initialized for the backup system as follows.

With IFS running, mount a backup pack on the extra drive. Connect to IFS from some other Alto using Chat, log in, enable, issue the Initialize command, and go through this dialogue:

! Initialize (file system type)

Answer 'Backup'.

Do you really want to create a file system?

Answer 'y'.

Number of disk units:

Answer '1'.

Logical unit 0 = Disk drive:

Type the physical unit number only of the drive on which the backup pack is mounted.

File system ID:

Type some short name that may be used to uniquely identify the pack, e.g., 'Backup1', 'Backup2', etc. No spaces are permitted in this identifier. It should be relatively short, since you will have to type it every time you mount the pack. (You

should mark this name on the pack itself, also.)

File system name:

Type some longer identifying information, e.g., 'Parc IFS Backup 1', or 'Serial number xxxx', or something.

Directory size (pages):

Type RETURN. (The default of 1000 pages is plenty.)

Ok? [Confirm]

Answer 'y' if you want to go ahead, or 'n' if you made a mistake and wish to repeat the dialogue.

IFS now initializes the backup file system, an operation that takes about 2 minutes for a T-80 and 7 minutes for a T-300. The message 'Done' is displayed when it is finished.

11.3. Setting backup parameters

The next step is to set the backup parameters, an operation that generally need be done only once. Issue the Backup command to enter the backup system command processor (whose prompt is '*'), then the Change command. It will lead you through the following dialogue:

* Change (backup parameters)
Start next backup at:

Enter the date and time at which the next backup run is to be started, in the form '7-Oct-77 02:00'.

Stop next backup at:

Enter the date and time at which the next backup run is to stop if it has not yet completed, e.g., '7-Oct-77 05:00'.

Interval between backup runs (hours):

Type '24'.

Full file system dump period (days):

Enter the number of days between successive backups of existing files (the parameter *n* above). A good value is 30.

The backup system command processor is exited by means of the Quit command in response to the '*' prompt.

11.4. Normal operation

The following commands are used during normal operation. All of them require that you first Enable and enter the backup system command processor by means of the Backup command.

* Status

Prints a message describing the state of the backup system. It will appear something like:

Backup system is enabled and waiting.
Backup scheduled between 7-Oct-77 02:00 and 7-Oct-77 05:00
File system Backup1 is available to backup system.
73589 free pages.

'Enabled' means that at the appropriate time the backup system will start up automatically; the opposite is 'disabled'. The backup system becomes enabled when you mount a backup pack (see Mount command, below), and disabled when the backup system can no longer run due to come condition such as the backup pack being full.

'Waiting' means that the backup system is not presently running; the opposite is 'running'. When it is running (or has been interrupted in the middle of a backup run for whatever reason), it will display an additional message of the form:

Presently working on file *filename*

as an indication of progress (files are backed up in alphabetical order).

The last lines display the status of the current backup pack (assuming one has been mounted. If several backup packs have been mounted, they will all be listed.) The possible states are 'available', 'presently in use', and 'no longer usable'. In the last case, the reason for the non-usability is also stated, e.g., 'Backup file system is full'.

* Enable (backup system)
* Disable (backup system)

Enables or disables running of the backup system. If Disable is issued while the backup system is actually running, it will stop immediately (within a few seconds). These commands are not ordinarily needed, because an Enable is automatically executed by Mount (see below) and a Disable is executed when the backup system finds that there are no longer any usable backup packs. The backup system also stops automatically if IFS is halted by the Halt command, but it is not disabled and will resume running when IFS is restarted.

* Mount (backup file system) *name*

Makes a backup pack known to the system. *name* is the file system ID of the backup pack (e.g., 'Backup1'). The pack must be on-line.

If the file system is successfully mounted, a message appears in the form:

Backup1 (Parc IFS Backup 1),
initialized on 6-Oct-77 19:32, 273 free pages.
Is this the correct pack? [Confirm]

If this is the pack you intend to use, you should answer 'y'. Then:

Do you want to overwrite (re-initialize) this pack? [Confirm]

Normally you will be mounting a backup pack that has either never been used before or was last used more than *n* (e.g., 30) days ago. In this case you should answer 'y'. This will cause the backup pack to be erased (destroying all files stored in it) at the beginning of the next backup run.

If, however, you are re-mounting a partially-filled backup pack that was removed for some reason, you should answer 'n'. The backup system will then not erase the backup pack but rather will simply copy additional files to it.

* Dismount (backup file system) *name*

Makes a previously mounted backup pack unavailable to IFS. This command may be issued only while the backup system is disabled (use the Disable command if necessary).

The normal operating procedure is very simple. Every day, issue the Enable and Backup commands to enter the backup system command processor, then issue the Status command. The status will indicate one of the following conditions:

1. 'Enabled and waiting', with one or more packs 'available to backup system'. In this case you need not do anything.

2. 'Disabled and waiting', with one pack 'no longer available to backup system' because 'Backup file system is full'. In this case, you should remove the backup pack, install another one (making sure it was last used more than *n* days ago), and declare it to IFS by means of the Mount command (above).

3. 'Disabled and waiting', with some other condition (e.g., 'Can't find logical unit 0'). You should correct the condition (most likely the required pack wasn't mounted at the time the backup system last started to run), then issue the Mount command as above.

When done, issue the Quit command to exit the backup system command processor. It is a good idea to keep a record of the dates on which each backup pack was mounted and dismounted so that you know when a pack is available for re-use.

11.5. Restoring individual files from backup; listing backup files

Individual files may be restored from backup in the following manner. It is not a good idea to do this while the backup system is running.

Install the desired backup pack on any free drive. Issue the Enable and Backup commands to enter the backup command processor. Then go through the following dialogue:

* Restore (from backup pack) *name*
name (*long-name*) mounted
Restore: *file-designator*

The *name* is the File system ID of the backup pack (e.g., 'Backup1'). In response to 'Restore:', type the name of a file to be restored. '*'s are permitted, and the default version is '!*'. The name of each file is typed out as it is restored.

When all files matching *file-designator* have been restored, IFS will again prompt you with 'Restore:'. You may either restore more files (from the same backup file system) or type RETURN to indicate that you are finished.

Files are restored from the backup system with precisely the attributes (version number, reference dates, etc.) they had when backed up. If a file already exists in the primary file system, IFS will refuse to overwrite it unless the version in the backup file system is *newer.*

It is also possible to examine the directory of a backup pack (or, indeed, any IFS file system you can mount in its entirety) by means of the following commands:

* OnLine (file system) *name*
name (*long-name*) mounted

Makes the secondary file system *name* available for use by the commands described below. If some other file system was already put on-line by a previous OnLine command, it is first put off-line.

* List (files) *file-designator*

This command is identical to the standard top-level List command (with all its sub-commands), but applies to the secondary file system most recently specified in an OnLine command rather than to the primary file system.

* OffLine

Makes unavailable the file system most recently specified in an OnLine command. This operation is performed automatically if you exit the Backup command by Quit or control-C.

After doing an OnLine, you may issue as may List commands as you want; control remains in the Backup command (unless you abort by control-C), and the secondary file system remains on-line. A Restore command will use the current secondary file system if there is one, and will automatically put it off-line when it is finished.

You *must* issue OffLine or an equivalent command before turning off the drive on which the secondary file system is mounted. Failure to do so will cause IFS to fall into Swat.

11.6. Reloading the entire file system from backup

If the primary file system is clobbered in a way that the Scavenger can't repair, the following procedure may be used to recreate it from backup. If performed correctly, this procedure will restore the primary file system to its exact state at the time of the most recent backup run.

11.6.1. Complete reload

First, re-initialize the primary file system as described earlier (section 4). Then connect to IFS from another Alto using Chat, login as System (password IFS), and issue the Enable command. It is advisable at this point to disable logins with the Disable Logins sub-command of the Change System-parameters command so as to prevent users from accessing the file system while you are reloading it.

Mount (on any free drive) the *most recent* backup pack, i.e., the one most recently written on by the backup system (this is very important). Then:

```
* Reload (file system)
Reload the entire file system? [confirm] yes
Note: mount the LAST backup pack first.
Mount backup pack: name
```

The *name* is the ID of the backup pack you have mounted. IFS will now proceed to restore files from the backup pack to the primary file system. When it is done, it will again ask you to 'Mount backup pack:', at which point you should mount the next most recent backup pack. Repeat this procedure until you have mounted all packs written within the past *n* days. When you are done, type control-C to terminate the reload process.

IFS will list out the files as they are restored. (To disable the pause at the end of each page, type ahead one space.) You will notice that not all files are restored. In particular:

Files that were backed up at some time but no longer existed at the time of the last backup are not restored. (The listing will say such a file is 'deleted'.)

Files already restored from a more recent backup are not restored from an earlier one. (The listing will say 'already exists'.)

It is important to understand the difference between Restore and Reload. Restore simply copies the specified files from the backup pack to the primary file system. Reload, on the other hand, attempts to recreate the state of the primary file system as of the most recent backup. To this end, Reload will restore only those files that actually existed at the time of the most recent backup run, and will

skip files that once existed (and were backed up) but were subsequently deleted.

It is *essential* that the last backup pack be reloaded first. Failure to heed this instruction will cause some files not to be reloaded that should have been, and vice versa. If the reload is interrupted for any reason and must be restarted, you must again start by reloading the last backup pack (even though all files from that pack may have been reloaded already), and *then* skip to the pack whose reload was interrupted. This is because the decision whether or not to reload each file is made on the basis of the last state of the file system as recorded on the most recent backup pack.

Reloading the file system restores all system and backup parameters to their former state, so long as the System directory is one of those restored. If you are using the backup system to move files *en masse* from one file server to another, you should check to make sure that the system parameters that are restored are suitable for the new system. Also, after completing a reload, it is necessary to halt and restart IFS to ensure that all system parameters take effect.

11.6.2. Partial reload

Ordinarily you should answer 'yes' to the question 'reload the entire file system?'. However, there are situations when you might wish to reload only some of the directories, such as when moving a group of users' files from one IFS to another. The easiest way (though not the only way) to accomplish this is to reload those directories from the original IFS's backup packs onto the new IFS.

If you answer 'no' to 'reload the entire file system?', IFS will ask you to type in the names of the directories to be reloaded, separated by spaces and terminated by RETURN. You should type these names carefully, as they are not being error-checked by IFS and there is no way to go back and make a correction. (Do not type angle brackets around the directory names.) After you type RETURN, the remainder of the reload process will take place as described above, but only the directories you specified will be restored and the rest will be skipped.

11.6.3. Reloading damaged files

When the IFSScavenger is run to repair a broken file system, it may find one or more files whose contents are damaged. The IFSScavenger is capable of repairing only the file system's structure, not its contents. When it detects a damaged file, it marks the file as being damaged, sets the file's protection to empty so as to discourage access by users, and puts an error report in the IFSScavenger.log file. (Certain kinds of damage cause the file to be deleted altogether; this is also noted in the log file.)

When only a few files are damaged or deleted, the easiest recovery procedure is to restore them individually using the Restore command described in section 11.5. But when many files are involved, it is better to use the following procedure, which is relatively automatic but quite time-consuming.

The procedure is simply to use the Reload command described in section 11.6.1, but without first initializing the file system. Reload will consider all the files on all the backup packs you mount (starting with the most recent one), but will copy only those files that are either damaged or missing in the primary file system. This procedure may be carried out while the file server is available to users.

Note that this operation will also reload copies of any files that were deliberately deleted by users since the most recent run of the backup system. This is because the Reload process has no way of determining whether a missing file was deleted by a user or by the IFSScavenger. After completing this procedure, you should warn users that recently-deleted files may have been resurrected.

11.7. Repeating backup runs

It may happen that you want to repeat one or more of the most recent backups—say, because the current pack suffered hard errors or irreparable damage, and you wish to repeat the backup of the affected files onto a fresh backup pack. This is controlled by the following commands:

* Repeat (backups later than) *date*

During the next run of the backup system, IFS will back up all files that were last backed up more recently than *date*, in addition to the files it normally backs up.

* Don't Repeat (backups)

Cancels the Repeat command. The Repeat command is also cancelled automatically upon successful completion of a backup run.

12. Accounting

Accountant.run is a program which collects accounting and administrative information from a running IFS. It retrieves copies of all of the Directory Information Files (DIFs) from a running IFS and produces a text file containing per-directory and system-wide information.

In order to run Accountant, you must be a wheel, since the DIFs which Accountant reads are protected. Note that you run this program on some *other* Alto, not on the IFS Alto.

When first started up, Accountant asks you for the name of the IFS that it is to connect to. It then asks three questions: 'Generate accounts listing?', 'Generate group membership summary?', and 'Generate disk usage listing?'; for each one, if you answer 'yes' then it requests an output file name (on your local Alto disk). It then connects to the IFS and produces the requested output.

An accounts listing consists of the names and attributes of all directories in the system, including disk page limit and current usage. At the end of the listing are the totals of page limits and current usages.

A group membership summary shows the memberships of each of the IFS user groups. This information is valid only for non-Grapevine members of non-Grapevine groups. The group membership summary is useful in managing an IFS that does not use Grapevine for access control, and is also useful when converting an existing IFS from non-Grapevine to Grapevine access control. Additionally, the summary includes, for each group, a list of directories whose connect, create, or default file protections refer to the group; this is useful in determining what a group is used for and whether it is still in active use.

A disk usage listing includes, for each directory, the number of files and pages accounted for by 'old' versions (all but the most current version of files for which multiple versions exist) and a histogram of time since most recent access. This information is useful for discovering obsolete files and directories.

The accounts listing and group membership summary are generated fairly quickly—15 minutes or so for a large IFS (4 T-300 disks). The disk usage listing takes a long time—2 to 3 hours for a large IFS. All three listings can be generated simultaneously; however, due to peculiarities of the FTP protocol, generating a disk usage listing at the same time as either or both of the others is likely to take longer than generating them separately.

13. Miscellaneous

13.1. Disk pack identification

If you forget the ID of some Trident pack (e.g., a backup pack), there is no way to 'Mount' it for the backup system. This is why it is a good idea to mark the ID on the pack itself (not on its plastic cover, which is interchangeable with other packs). A good place to mark it is on the plastic ring on the top of the pack. Do *not* affix a paper label: it will fly off and destroy the heads, the pack, or both.

There is, however, a command for finding out vital information about a pack. It is:

! What (is the pack on drive) *d*

where *d* is a drive number. If the pack is an IFS pack (primary or backup), this command will print out the vital parameters, including the ID. If the pack is not an IFS pack, it will say so.

13.2. Software performance

The IFS software strains the Alto's capacity severly, particularly with respect to main memory. In combination with certain deficiencies of the BCPL runtime environment, this can lead to rather poor performance (in particular, excessive disk thrashing) when there are more than a few simultaneous users of the system.

Also, there are times when certain data structures and code segments cannot be swapped out. It is possible for the system to deadlock if all of memory is occupied by such immovable objects. The symptom of this is that IFS ceases to respond to requests for service, the Alto screen looks normal (black with 'IFS' in the cursor), and the screen does not flash when you press the space bar. The possibility of deadlocks is the principal reason for imposing a limit on the number of simultaneous server processes. To make things worse, deadlocks frequently occur during major changes to the directory, thereby leaving it in an inconsistent state requiring the IFSScavenger to correct.

If the IFS Alto has only 64K of memory, IFS must both keep data and swap code using that memory. Considering that there is over 100K of swappable code and only 32K of memory available for swapping both code and data, this leads to serious disk thrashing. In the 64K configuration, the maximum possible Server-limit is 6 and the default is 5. Even with a limit of 5, memory deadlocks are likely (depending on usage patterns), and it may be necessary to reduce the Server-limit to 4 or even 3 in order to entirely prevent deadlocks. For all practical purposes, a 64K Alto should be considered an unsupported configuration.

If the IFS Alto has extended memory, the situation is much better. IFS has an 'extended emulator' that is able (with some restrictions) to execute BCPL code residing in extended memory, though it can still reference data only in the first 64K of memory. Consequently, performance is significantly improved, since most or all of the first 64K is available for data and code swapping is reduced or eliminated.

The IFS software running on an Alto with 128K of memory can support up to 8 simultaneous users, and with 192K or more up to 10 simultaneous users. It is believed that memory deadlocks are impossible with these configurations. Therefore, it is strongly recommended that IFS Altos have at least 128K of memory, and systems that serve large or highly demanding user communities should have 192K of memory. (The software does not presently take any advantage of memory beyond 192K.)

13.3. Interpreting system statistics

The IFS Executive's Statistics command pours out various internal operating statistics, some having to do with hardware and some with software. Many are of interest only to IFS implementors, but all are explained here for completeness. An IFS administrator should examine these statistics periodically (say, once per day) to notice problems that may lead to progressive failures; this is particularly important in the case of memory and disk errors. Except where noted, all statistics cover the interval since IFS was last restarted.

If you terminate the Statistics command with SPACE rather than CR, you will be asked for an additional keyword (one of Directory, Disk, Mail, Memory, or Server); and only the specified subset of the system statistics will be displayed.

SmallZone overflows, bigZone overflows, overflow pages

> IFS has a three-level memory storage allocator. SmallZone and bigZone are heap-type allocators for objects of less than 25 words and of 25 to 500 words, respectively. Objects larger than 500 words are allocated by removing one or more 1024-word pages from the VMem (virtual memory manager) pool. If one of the first two zones becomes exhausted, it recovers by borrowing space from the next larger zone.
>
> It is normal to encounter up to 100 or so zone overflows per day, and for there to be a maximum of 2 or 3 VMem pages used to recover from bigZone overflows. More overflows are indicative of the need to change some compile-time parameters. If the 'current' number of overflow pages remains nonzero for any significant length of time, it is indicative of a bug (loss of allocated storage).

Net blocks allocated minus blocks freed

> This is simply the number of memory storage blocks presently allocated. If there is no system activity besides your Chat connection, this should be more-or-less constant. If it increases slowly over time, storage is being lost.

PBIs, PBI overflows

> The Pup software maintains a pool of packet buffers (PBIs) that are shared among all active servers. The first number displayed is the normal number of PBIs, which is constant for a given IFS release and Alto memory configuration. Under certain circumstances (particularly when connections are made through slow links), the system runs out of PBIs; when this happens, additional PBIs are temporarily allocated (from bigZone) to cover the shortage. (Frequently this will cause bigZone to overflow as well.)

VMem buffers, buffer shortages

> Approximately half of Alto memory is turned over to the VMem package, which manages it as a buffer pool of 1024-word pages and implements a software virtual memory for accessing various objects on the disk, including code overlays (if not resident in extended memory), directories, and bit tables. The number of VMem buffers is constant for a given IFS release and Alto memory configuration.
>
> If the VMem package receives a request that it can't satisfy because all buffers are in use by locked objects (or have been removed to service a zone overflow), it increments the 'buffer shortages' count (*vMemBufferShortages*, accessible from Swat) and then waits, in the hope that some other process will run to completion and release some buffers. Sometimes this works. On other occasions, all processes in the system get into this state and the system is deadlocked.

VMem reads and writes

> This table contains the number of swap reads and writes for each of four main types of

objects managed by the VMem package: code overlays, VFile pages (virtually accessed files, principally the IFS directory B-Tree), DiskDescriptor (disk bit map) pages, and Leaf pages (file pages accessed via the Leaf server).

Overlays read from XM and from disk

If the Alto has extended memory, this indicates how many overlay reads have been satisfied by reading from extended memory rather than from the disk. Most overlays are executed directly by the extended emulator, but under certain conditions overlays must be swapped into the first 64K before execution. There should be virtually no overlay swapping (from either XM or disk) on a machine with 192K of memory or more.

Main memory errors

For an Alto-II, if any main memory errors have occurred since IFS was last restarted, the offending memory card and chip numbers are reported. These are single-bit errors that have been corrected by the hardware, so they are not cause for immediate alarm. However, when such errors occur, you should schedule hardware maintenance for replacement of bad chips at the next convenient time. This will reduce the likelihood that a single-bit error develops into an uncorrectable error, causing the server to crash.

Disk statistics (cumulative)

This table contains operating statistics for each Trident disk unit, and, in the case of T-300 disks, each of the two logical file systems on the unit. All disk statistics are cumulative from the time the file system was created.

File system	File system name and logical unit number within that file system. Logical unit 0 contains the IFS directory and the code swapping region.
Transfers	Number of pages transferred to and from the unit.
Err	The number of errors of all kinds except data-late errors and label check errors knowingly caused by the software. (See below for more information.)
ECC	The number of data errors detected by the Error Correction Code. (See below for more information.)
Fix	The number of ECC errors that have been corrected. The ECC permits correcting error bursts up to 11 bits long.
Rest	The number of times a head-restore operation has been performed. This is done as a last-resort measure when an uncorrectable error persists through 8 retries.
Unrec	The number of errors unrecoverable after 16 retries. This usually causes IFS to fall into Swat and report an unrecoverable disk error. This may be indicative of a hardware problem or of an inconsistency in the structure of the file system. Running the IFSScavenger can tell you which.
BTerr	The number of times the bit table has been found to be incorrect, i.e., to claim that a page is free when it isn't. This in itself is a non-fatal error, but it may be indicative of serious hardware or software problems. On the other hand, it can also be caused by restarting IFS after a crash without first running the IFSScavenger.
Free	The number of free pages on the logical unit. IFS always allocates new files on the emptiest unit, and every file is contained completely within one unit. The software does not always recover from running out of disk space

(i.e., it may fall into Swat), so be careful not to let the amount of free space get too low.

Disk drive statistics (since last restart)

This table corresponds to a portion of the previous table, but it is reset every time IFS is restarted, and the software-related information is omitted. Progressive hardware problems are more evident in this table than the previous one.

Transfers — Number of pages transferred to and from the unit.

Err — The number of errors of all kinds except data-late errors and label check errors knowingly caused by the software. It is normal for these to occur at a low rate; they are caused by known hardware problems in the controller and disk drives, and by random electronic glitches. A sudden jump in this error count *not* accompanied by a corresponding jump in the ECC count is grounds for suspicion of a non-data-related problem (e.g., positioning errors or momentary not-ready conditions.)

ECC — The number of data errors detected by the Error Correction Code. These should be extremely infrequent, especially on T-300 drives which are very reliable if properly maintained. A sudden jump in the rate of ECC errors is grounds for suspicion of a hardware problem.

/10↑10 bits — The ECC error rate per 10^{10} bits transferred. The Century Data specification for T-300 drives is one recoverable ECC error per 10^{10} bits; but this assumes perfect testing of disk packs, no interchanging of packs between drives, etc. Nevertheless, this statistic is useful for comparison between drives in one system, or between different systems.

Directory statistics

These include the total number of files in the system; the number of pages actually in use by the directory B-Tree; and (most important) the number of runs (fragments) comprising the directory file <System>IFS.dir. Ordinarily there should be only one run; however, if the directory has grown larger than its preallocated size (as discussed in section 10), the additional required pages will cause more runs to be created. (This can also occur if IFS.dir suffers hard disk errors and is destroyed and recreated by the IFSScavenger.) If the number of runs exceeds 40, IFS must be started with one or more /F switches, as described in section 4; otherwise it will crash occasionally due to running out of space in its file map.

Server statistics

These show, for each type of server, the number of connection requests that have been accepted and the number that have been refused due to the system reaching the Server-limit.

Mail statistics

These are described in section 8.1. They are cumulative since the file system was created or the mail statistics were last reset explicitly.

14. Known bugs and what to do about them

No bugs are presently known to exist in IFS version 1.37.

15. Revision history

IFS was designed during November and December of 1976. It was storing and retrieving files in January 1977, and was 'released to friends' in June of 1977 (releases through 1.02). The first 'official' release to the Alto user community was:

Version 1.03; August 4, 1977

> Procedures added for running Triex, for blessing your Trident controller, and for halting IFS in a cleaner manner than before.

Version 1.07; September 3, 1977

> /V switch added for startup-time directory B-Tree verification.

Version 1.08; October 5, 1977

> Backup system released.

Version 1.10; November 1, 1977

> Full T-300 support; 'Extend' command for adding packs to an existing file system; Triex and TFU operating procedures changed; IFSScavenger released.

Version 1.12; November 10, 1977

> Performance improvements in both IFS and IFSScavenger; procedures for initializing and testing a disk pack again changed (Triex eliminated from procedure).

Version 1.14; February 21, 1978

> File protections implemented; command added to disable logins; backup system bugs fixed.

Version 1.15; March 4, 1978

> Converted to new time standard; 'What' command added; automatic SetTime at startup; obscure directory ordering bug fixed.

Version 1.18; November 15, 1978

> Mail server added; limited support for extended memory; Change System-Parameters command added, with sub-commands to change clock correction, limit the number of simultaneous servers, reset the time, and enable and disable service; Logins command removed; screen flashes if you hit space bar and system is operating normally; Accountant program released; documentation on system performance and interpreting Statistics output; file IFS.Ov is no longer part of the release.

Version 1.21; July 16, 1979

> Mail forwarding and Press file printing implemented; miscellaneous servers (name, time, and boot) added; Change System-Parameters sub-commands modified; protection groups may now be 'owned' by individual (non-wheel) users; Change Group-Membership and Show Group-Membership commands added to permit users to manipulate group membership; more statistics.

Version 1.23; January 13, 1980

> Conforms to new file creation date standard; files-only directory's owner is permitted connect access; mail server supports remote distribution list retrieval; privileged Telnet Mail command added, which brings together the mail related commands previously scattered

among other Telnet commands.

Version 1.24; March 8, 1980

Extended emulator included, enabling substantially improved performance and more simultaneous users if the IFS Alto has extended memory (see section 13.2); new command file available to construct an IFS Operation disk from scratch (section 2); a few additional statistics are kept and displayed by the Statistics command (section 13.3); sub-command added to privileged Mail command to reset the mail statistics (section 8.1); optional Leaf page-level access server included on a 'use at your own risk' basis (section 13.4). Note: due to a format change, the mail statistics will be reset automatically the first time IFS 1.24 is started.

Version 1.25; May 19, 1980

CopyDisk server added (see section 13.5); new commands to display and cancel press printing requests (section 7); new commands to repeat the backup of files (section 11.7).

Version 1.27; September 6, 1980

Mail server changes, required for compatibility with the Grapevine servers; Printed-by sub-command added to Print command; a few bugs fixed.

Version 1.28; January 3, 1981

Rename command defaults new file name; Print command has new sub-commands Duplex and Password; backup system can reload selected directories rather than the entire file system (section 11.6); boot server can now boot-load microcode for Dolphins and Dorados; the boot server's automatic acquisition of new boot files can be disabled (section 9); Accountant program augmented to produce disk usage listing (section 12); mail server changes, required for compatibility with the Grapevine servers (section 8.1). Note: the Leaf protocol has undergone minor changes that may require Leaf clients to be changed correspondingly; implementors of software that uses Leaf should contact Ted <Wobber.PA> for information.

Version 1.30; January 28, 1981

The Change Protection command has been generalized to Change Attributes, with new sub-commands Backup, Byte-size, and Type. The Backup command has new sub-commands OnLine, OffLine, and List (section 11.5). Some hardware error checks have been added: the Control RAM and S-registers are tested during startup (section 5), and corrected single-bit main memory errors are recorded (section 13.3). Additionally, a number of bugs have been fixed.

Version 1.31; May 10, 1981

The action of the /A switch has changed. The directory statistics (section 13.3) and the meaning of the /F switch are now documented. Mail system changes have been made to facilitate conversion of an IFS-based registry to Grapevine; in particular, disabling a user's mail capability causes his in-box to be destroyed next time it is read rather than immediately; and the forwarder now understands about registry names that map to multiple addresses (section 8). The Leaf server now supports a 'multipleWriters' mode of access; consult Ted <Wobber.PA> for details. The FTP server now deals in date properties that include time zones; in conjunction with a new release of FTP.run, this enables file date properties to be transferred correctly between different time zones.

Version 1.33; June 29, 1981

The purpose of this release is principally to fix two long-standing and notorious bugs: the 'B-Tree delete bug' and the 'infinite return-to-sender bug'. Additionally, the Trident disk

software's handling of recoverable disk errors has changed somewhat; in particular, the rate of non-ECC errors is substantially reduced. Some additional disk error information is displayed by the Statistics command (section 13.3).

Version 1.35; December 11, 1981

IFS can be configured to use Grapevine for authentication and access control; this replaces the user name and group structure maintained by IFS, and eliminates the need for Guest accounts (section 6). You should read the new 'How to use IFS' document, as it includes an explanation of how IFS uses Grapevine that is not repeated here. A Show System-Parameters command has been added. The Create and Change Directory-Parameters commands now have the same set of sub-commands (section 6). Reloading the file system from backup now properly restores all system parameters instead of resetting them to default values (section 11.6). Accountant generates a group membership summary (section 12). A new version of the IFSScavenger accompanies this release. Note: for proper error reporting, you should obtain the latest [Maxc2]<Alto>Sys.errors.

March 14, 1982 (documentation update only)

A summary of known bugs has been added (section 14). There is now a separate document describing access control policies and procedures in considerably more detail; please obtain and read [Maxc]<IFS>AccessControls.press.

Version 1.36; May 13, 1982

This is principally a maintenance release to fix a number of bugs found in previous releases. Functional changes are as follows. In an IFS that uses Grapevine group checking, a user who is *not* registered in Grapevine but *does* have a login directory on the IFS is no longer automatically a member of World; his membership in World is now controlled just the same as membership in other groups (using the Change Group-Membership command). The Backup Reload command may now be used to repair damage detected by the IFSScavenger (section 11.6.3). The FTP server supports some recent extensions to the FTP protocol that permit substantially improved performance in certain operations (particularly enumerations, which in certain cases are now over 10 times as fast as before); some changes to client software are required to take full advantage of this improvement.

Version 1.37; October 3, 1982

This release introduces some minor new features. The boot server can now boot-load Sun workstations; also, new boot files installed by manual means (e.g., FTP) are now noticed immediately instead of after a delay of up to 8 hours (section 9.2). A new server, LookupFile, is included and may optionally be enabled (section 9.6). The backup system now automatically fixes any incorrect directory page usage totals which it encounters. Additionally, internal changes in the VMem software have resulted in a modest performance improvement and elimination of the long-standing 'Can't flush locked page' bug.

Date: 29 July 1982 5:34 pm PDT (Thursday)

From: Boggs.pa

Subject: IFS Scavenger

To: IFSAdministrators↑

Reply-To: Boggs.pa

A new version of the IFS Scavenger is now available. Retrieve:
[Indigo]<IFS>IFSScavenger.run
[Indigo]<IFS>IFSScavenger.syms.
This is a maintenance release; there are no documentation changes. When it starts it should say "IFS Scavenger of July 27, 1982". Three things were changed:

1) When the Scavenger has to create or extend a critical system file (e.g. IFS.Dir), it now does it using the minimum number of page runs each as large as possible. This should eliminate the following problem: suppose a hard disk error causes the Scavenger to truncate or delete IFS.dir. Further assume that the file system is old (i.e. the free pages are scattered all over the pack) and that the file system is nearly full (disk usage expands to fill the available space). Before this change, the recreated directory file would consist of zillions of 1 or 2 page runs. This would run IFS's file map out of space (causing wierd crashes) unless you always started IFS (and the Scavenger) with several /Fs.

2) When the Scavenger finds a damaged file, it sets a bit in the file's leader page. LISTing this file from Chat will then display ** Damaged ** after the filename. A file system RELOAD (a backup system option) will automatically restore damaged files from backup. This feature was added to IFS 1.36, but I was busy doing other things at the time and didn't get around to putting the damage marking logic into the Scavenger until now.

3) the Scavenger no longer prints "[1-3] Inaccessible page nnn" messages unless the debug flag is set. If a big file (like IFS.Dir) got clobbered, these messages (one per page, along with some other info) often ran the model 31 disk (where the log is kept) out of space, causing other more important error messages to be lost.

While I have your attention: Ed Taft is on vacation until at least 17 Aug, so if you need IFS help call ME at 8*923-4421.

/David

Date: 28 Nov. 1982 1:25 pm PST (Sunday)

From: Taft.PA

Subject: IFS 1.37

To: IFSAdministrators↑

Reply-To: Taft

Version 1.37 of the IFS software is released. It has been running for over 6 weeks on Ivy and Indigo and for shorter intervals at several alpha-test sites with no unsolved problems.

Changes since IFS 1.36 are as follows. The boot server can now boot-load Sun workstations; also, new boot files installed by manual means (e.g., FTP) are now noticed immediately instead of after a delay of up to 8 hours. A new server, LookupFile, is included and may optionally be enabled. The backup system now checks the disk usage total of each directory, and fixes it if it is incorrect. Additionally, internal changes in the VMem software have resulted in a modest performance improvement and elimination of the long-standing "Can't flush locked page" bug.

Several bugs turned up during alpha testing and have been fixed. It was possible to hang the system by changing backup parameters at just the wrong time (this was a very long-standing bug). The mail server had stopped working altogether (I suspect it didn't work in IFS 1.36 either; more on this below). There was a very low-probability bug in Rename which caused occasional flakey behavior during heavy use (e.g., Brownie moving files en masse from one directory to another); this bug dates from IFS 1.35, and has had two symptoms: file system inconsistencies such as having two directory entries for the same file, and occasional crashes after Rename.

Complete information may be obtained from the revised "IFS Operation" document, which is [Maxc]<IFS>Operation.press or [Indigo]<IFS>Operation.press. The "How to use IFS" document is unchanged.

Software may be obtained from either [Maxc]<IFS> or [Indigo]<IFS>1.37>, and consists of files IFS.run, IFS.syms, and IFS.errors. For correct error reporting, please be sure you have the latest Sys.errors, which may be obtained from [Maxc]<Alto> or [Indigo]<Alto>. (Alpha-testers: if you are running IFS 1.36.10, you should convert to IFS 1.37 at this time.)

While I am on the subject, I should mention that we would like to discontinue support for the IFS mail server altogether in the near future. There are only a very few sites which have not yet converted to Grapevine and are relying on IFS mail servers. Maintaining this software, and maintaining the Grapevine software that provides compatibility with the old MTP protocol used by IFS, is a burden which is no longer justified by the amount of use this software receives.

Therefore, sites which are presently using IFS for mail service should begin planning to install a Grapevine server or to make arrangements to keep local

mailboxes on some existing Grapevine server.

Inter-Office Memorandum

To	IFS and Grapevine administrators	Date	March 13, 1982
From	Ed Taft	Location	PARC/CSL
Subject	Introducing new access control policies	File	[Indigo]<IFS>AccessIntro.bravo

XEROX

Introduction and motivation

The attached memo describes some new access control policies and procedures that are designed to improve information security in the Xerox Research Internet.

Generally speaking, the need for such policies is a consequence of the growth of the Internet to encompass a large number of diverse organizations. The specific reason for introducing these policies at this time is that foreign affiliates (Rank Xerox and Fuji Xerox) are becoming connected to the Internet; as a consequence, information transfer within the Internet is now subject not only to Xerox security guidelines but also to U.S. Government regulations.

These policies and procedures have been developed by a committee consisting of Andrew Birrell, Jerry Elkind, Mike Schroeder, and Ed Taft. We welcome any constructive criticisms or suggested improvements.

Implementation

The new policies call for some substantial changes to existing practices, particularly with regard to assignment of individual R-Names and proper use of groups for access control. Naturally, we don't expect that it will be possible to put them all in place immediately. The most urgently-required measures are the following:

1. Elimination of "Guest" and other individual R-Names with widely-known or easily-guessed passwords, in both IFSs and Grapevine.
2. Conversion of all remaining IFSs to use Grapevine for authentication and access control. (This is in progress, but is by no means complete.)
3. Registration of all individuals (particularly foreign affiliates) in their proper registries.
4. Elimination of "interest" groups and groups including non-affiliates and foreign affiliates from the set used for IFS access control.
5. Education of all users of the Internet in proper use of access controls so as to fulfill the information security requirements.

In particular, item (1) is of such crucial importance that we must request all administrators to begin action on it immediately.

In some cases, compliance with the new policies will require changes to be made to existing software; this is particularly true of software that uses compiled-in credentials (a problem discussed in considerable detail in the attached memo). Since management approval may be required for the implementation of such software changes, administrators should see that this memo is brought to the attention of appropriate levels of management.

Date: 5 June 1982 3:20 pm PDT (Saturday)

From: Taft.PA

Subject: IFS 1.36

To: IFSAdministrators↑

Reply-To: Taft

Version 1.36 of the IFS software is released. It has been running for over a month on Ivy and Indigo and for two weeks on three other servers with no unsolved problems. This is likely to be the last IFS release for a very long time.

This is principally a maintenance release to fix a number of bugs found in previous releases. All known bugs have been fixed with the exception of the very rare "Can't flush locked page" bug, which we have decided not to try to fix. Functional changes are as follows:

1) In an IFS that uses Grapevine group checking, a user who is NOT registered in Grapevine but DOES have a login directory on the IFS is no longer automatically a member of World; his membership in World is now controlled just the same as membership in other groups (using the Change Group-Membership command).

2) The Backup Reload command may now be used to repair damage detected by the IFSScavenger. (Note: the version of IFSScavenger that supports this is not yet released. In the absence of IFSScavenger support, the IFS Backup Reload command will restore missing files but will not replace damaged ones; you must first delete damaged files manually, using the IFSScavenger.log as a guide.)

3) The FTP server supports some recent extensions to the FTP protocol that permit substantially improved performance in certain operations (particularly enumerations, which in certain cases are now over 10 times as fast as before); some changes to client software are required to take full advantage of this improvement. (The protocol extensions are described in the revised FTP specification, filed as [Maxc]<Pup>FTPSpec.press.)

The software is available from the usual place:

[Maxc]<IFS>IFS.run
[Maxc]<IFS>IFS.syms
[Maxc]<IFS>IFS.errors

The "How to use IFS" and "IFS Operation" documents have been revised; please obtain:

[Maxc]<IFS>HowToUse.press
[Maxc]<IFS>Operation.press

Date: 12 May 1982 3:49 pm PDT (Wednesday)

From: Taft.PA

Subject: "Can't flush locked page"

To: IFSAdministrators↑

Reply-To: Taft

There is a long-standing software problem that has been around since the first release of IFS over 5 years ago. The symptom is that the server falls into Swat with the error:

CallSwat from xxxxxx
Can't flush locked page

This bug is reasonably well understood but extremely difficult to fix. It's my impression that the bug strikes so rarely as not to be worth the effort required to fix it. (For example, I don't believe this has happened more than about once a year on Ivy and Indigo.)

If this bug strikes your IFS more frequently than once a year, I'd like to hear about it.

Ed

Inter-Office Memorandum

To	IFS and Grapevine administrators	Date	March 13, 1982
From	Ed Taft	Location	PARC/CSL
Subject	Access controls	File	[Indigo]<IFS>AccessControls.bravo

XEROX

In recent years, the Xerox Research Internet has grown to encompass a large number of organizations, including some outside the United States. Because of this, it is no longer possible to ignore the necessity of controlling access to electronic information resources within the Internet.

In this memo, we outline the information security requirements that must be met, and then describe the procedures for achieving them by means of the IFS and Grapevine access control mechanisms.

This discussion assumes a world in which all users are registered in Grapevine and all IFSs use Grapevine for authentication and access control. This is not yet the case. Organizations that have not yet converted are strongly encouraged to do so, since fulfilling the information security requirements is difficult or impossible without the mechanisms provided by Grapevine.

While this memo focusses on information stored in files on IFSs, much of the material is of more general relevance and applies to electronic and non-electronic information of all kinds.

1. Xerox information security requirements

Within Xerox, information is classified into five categories:

1. *Public-domain* information—that which has been formally cleared for public release outside Xerox.
2. *Proprietary* information—all information not cleared for public release but not included in one of the following more restricted categories.
3. *Private* data—information whose unauthorized disclosure could have a substantial detrimental effect on the operations of the company.
4. *Registered* data—information whose unauthorized disclosure could cause serious damage to the operations of the company.
5. *Personal* data—information of a sensitive, personal nature.

Information in the last two categories is subject to very stringent regulations on how it may be disseminated and stored. Since relatively few people have occasion to deal with registered and personal data, we shall concentrate on the other three categories.

For each category, there are guidelines for how the flow of that information should be controlled. Some of these guidelines are intended to protect Xerox proprietary concerns, while others are imposed by U.S. Government regulations.

For the purpose of describing appropriate degrees of access to Xerox information, we divide the universe of people into four groups:

1. *U.S. employees and affiliates*—U.S. citizens and permanent residents who are employees of Xerox organizations and subsidiaries in the U.S., and other U. S. citizens or permanent residents who have signed non-disclosure agreements with Xerox. This also includes employees of Xerox organizations in Canada.

2. *U.S. non-affiliates*—U.S. citizens and permanent residents not in category 1.

3. *Foreign affiliates*—employees of Xerox subsidiaries outside the U.S. Strictly speaking, this category also includes foreign nationals working in the U.S. under a temporary visa.

4. *Foreign non-affiliates*—foreign nationals not in category 3.

Persons in each of these groups are permitted access to categories of information on the following basis:

1. U.S. employees and affiliates:

   a. may have unrestricted access to *public-domain* and *proprietary* information;

   b. may be given access to *private* data on a need-to-know basis.

2. U.S. non-affiliates:

   a. may be given access only to *public-domain* information.

3. Foreign affiliates:

   a. may have access to *public-domain* information;

   b. may have access to *proprietary* information on a per-project basis only; project-wide approval by the International Deputy is required (see section 5.1), and information transfers *require* Export Control Coordinator approval and any other approvals that the organization "owning" the information decides are appropriate;

   c. may be given access to *private* data on a need-to-know basis; project-wide approval by the International Deputy is required, and information transfers *require* Export Control Coordinator approval (see section 5.1) and the other approval associated with private data information;

   d. A record must be kept by the Export Control Coordinator of all transfers of non-public-domain information to foreign affiliates.

4. Foreign non-affiliates:

   a. may be given access only to *public-domain* information;

   b. must not individually be on the U.S. Government denial list or members of organizations or countries on this list.

   c. A record must be kept of all transfers of information to foreign non-affiliates except Canadians; this includes *all* technical information, even though in the public domain.

2. Organization of access controls

Now we review the mechanisms that exist for controlling access to electronic information in the Xerox Research Internet.

To begin with, it should be understood that the Internet is designed to permit any connected machine to communicate with any other, without any controls or restrictions. Since the Internet extends outside the U.S., this enables unrestricted international communication. Any required restrictions on information transfer are the responsibility of the end parties of the communication, not of the Internet.

2.1. Basics of access control

The principal means of ensuring that a certain piece of information can be accessed only by certain individuals is by attaching an *access control list* to the information and by requiring that individuals be *authenticated.* Let us consider what this means.

An access control list is simply a list of names of individuals who are to be granted access to the associated information. For example, a file stored on a file server has a *protection* which, simply put, is an access control list that determines who may access the file. Upon each attempted access, the file server checks the name of the individual requesting access against the file's access control list, and permits the operation to proceed only if a match is found. This is entirely the *server's* responsibility, though the server can delegate some of the work to other servers as will be discussed shortly.

In order for this style of access control to be effective, it is necessary for the server to be able to determine that an individual's *name* actually represents that individual. This is the purpose of the *password.* The name and password together serve to *authenticate* the individual—that is, to identify the user and to verify his authenticity by requiring him to provide some piece of information that only he knows.

It should be clear why it is vital that users choose passwords with care and keep them secret. In an access control list based protection system, access is granted solely on the basis of *who the requestor is,* and not on criteria such as the requestor's physical location, ability to supply a password for the information being accessed, or other credentials that the user might possess. An individual's name and password is intended to represent *that individual* and nobody else.

2.2. Individual names and authentication

In the Xerox Research Internet, an individual is identified by a Grapevine *registered name* or "R-Name" composed of two parts, a *simple name* and a *registry,* separated by a period. A registry is a logical grouping of names, usually on a geographical or organizational basis; and a simple name identifies a specific individual within the registry. Examples of R-Names are "Smith.PA" and "Jones.EOS". A complete R-Name uniquely identifies one individual. In the Xerox 8000-series products, R-Names are composed of three parts instead of two, but otherwise are organized essentially the same.

The Grapevine servers maintain, for each individual, a password and various other attributes. One of the services provided by Grapevine is to authenticate an R-Name and password.

When a user requests service from, for example, a file server, that server first demands that the user (or client program acting on his behalf) provide a valid R-Name and password, which it asks Grapevine to authenticate. This process of "logging in" serves solely to identify the user; by itself it confers no access rights. That is why any authenticated user is permitted to "log in" to any IFS. Control over the user's access to information is accomplished by an entirely separate mechanism.

2.3. Groups and access control

Grapevine also maintains *groups.* A group, to first order, is simply a list of R-Names. A group itself is identified by an R-Name (which customarily, though not necessarily, contains a "↑" character).

Groups are used as access control lists, as well as for other purposes such as directing the distribution of messages. If a group is attached to some piece of information as its access control list, then an individual can access that information only if his R-Name is included in the group. This is the fundamental basis for access control in IFS, as well as in Grapevine itself.

Groups can contain other groups; this capability can be used to model organizational hierarchies, project membership, and various other structures. By using an appropriate group name for access control, information may be made available to every member of an organization, project, etc., even though the actual membership of that organization or project changes over time.

Groups can also contain *patterns* such as "*.PA"; any individual whose R-Name matches the pattern is considered a member of the group. This facility is provided as an administrative convenience in defining all-encompassing groups (and avoiding the need for exhaustive enumeration); but it does have certain consequences that will be discussed later.

Permission to change the membership of a group is itself controlled by access control lists. Some groups may be changed only by duly authorized managers of the organizations or projects which they represent. Other groups (the so-called "interest lists") permit any individual to add or remove his own R-Name.

The way this works is as follows. Each group has two access control lists called the *Owners* and *Friends* lists. An individual whose R-Name is in the Owners list is permitted to change the membership of the group arbitrarily (as well as to perform certain other operations). An individual whose R-Name is in the Friends list is permitted only to add or remove his own R-Name in the group's membership list. Centrally controlled groups have an empty Friends list, whereas completely uncontrolled groups have a Friends list of "*", a pattern that matches any R-Name.

2.4. IFS file protections

The preceding section described the general use of Grapevine groups as access control lists. The actual use made by IFSs is somewhat more complex.

Each file stored on an IFS has a *protection* consisting of two access control lists; roughly speaking, one controls reading and the other writing. Actually, there is a third list controlling appending, but that is of no relevance to the present discussion. Additionally, each directory on an IFS has a default file protection, which is applied to newly-created files in that directory. Finally, each directory also has two access control lists that control permission to *create* new files in the directory and to *connect* to the directory for the purpose of performing owner-like operations such as changing the access control lists themselves.

The group R-Names that may be mentioned in these access control lists are limited to a relatively small set that is chosen by the IFS's administrator. Thus it is the administrator's responsibility to ensure that only suitable groups are used as access control lists.

Each IFS also has a special access control list called "World" which represents some all-encompassing user group. For IFSs in the U.S., "World" is usually defined to be USRegistries↑.internet, which consists of all registered Xerox employees and affiliates in the U.S.

The intent of this arrangement is that "World" be included in the access control lists of all files that are *proprietary* but are not in a more restricted classification (such as *private* or *registered*) and have no other reason for more limited access. This facilitates communication among Xerox employees. Foreign nationals (whether or not they are Xerox affiliates) are denied access to such files in conformance with the information security requirements presented in section 1.

3. IFS and Grapevine administrative policies

In this section we describe specific IFS and Grapevine administrative policies that are intended to ensure that the information security requirements are fulfulled. Note that these policies must be applied consciously by the administrators; they are not enforced automatically by the software.

3.1. Registry membership

Each Grapevine registry is typically maintained by a single person or a small group of specially-designated people. The registry maintainer has the responsibility for ensuring that only valid individuals are registered.

First of all, it is important to understand that there are two classes of registries: those that contain human individuals (and groups of individuals) and those that contain other names used for special purposes. All of the familiar registries belong to the first class, such as PA, ES, Wbst, etc. Registries in the second class contain individuals that do not represent human users but rather machines, programs, or processes; for example, the GV and MS registries, used for internal control over the Grapevine data base, belong to this class. These two classes of registries must not be confused, for reasons that will become apparent shortly.

With this detail behind us, we now state the first principle of Grapevine registry administration:

1. Every individual R-Name registered in a normal organizational or geographical registry must correspond to a human user; and the R-Name is for the exclusive use of that user.

This rules out assigning individual R-Names for "guest" or communal use or for "automatic" access by programs that have such R-Names compiled into them. (This constitutes a major break with past policy. Situations for which such ficticious R-Names have been assigned in the past may be dealt with by the procedures described in section 4.)

2. Every individual registered in a Xerox U.S. registry must be a U.S. employee or U.S. affiliate (i.e., a member of group 1 in the classification presented earlier).

That is, every individual in registries such as PA, ES, and Wbst must be a U.S. Xerox employee or affiliate. Non-affiliates and foreign affiliates must be segregated into separate registries. For example, there exist registries RX and FX containing members of the Rank Xerox and Fuji Xerox organizations. It is straightforward to create new registries for other categories of individuals.

The reason for this is that the registries themselves constitute groups whose names are the patterns "*.PA", "*.ES", etc; all individuals in these registries are members of their respective groups. The group USRegistries↑.internet is defined in terms of these registry groups; its present composition is: *.DLOS, *.EOS, *.ES, *.Henr, *.LB, *.PA, *.STHQ, *.Wbst, *.XRCC. Since this is intended to describe all U.S. Xerox employees and affiliates, it is clear that individuals who are not U.S. Xerox employees or affiliates must not be assigned to these registries.

3.2. Group classification

There are three basic classes of Grapevine groups, characterized by their style of use:

1. Organization groups, which reflect the corporation's organizational hierarchy. Each individual who is a Xerox employee is ordinarily a member of exactly one organization group corresponding to that individual's immediate organization. That group is in turn a member of some larger group; and this structure continues up to the root of the hierarchy. An organization group may be modified only by administrators associated with the organization, to reflect new hires, terminations, and transfers.

2. Project groups, composed of members of individual projects. These frequently cut across organizational boundaries, and may have a hierarchical structure of their own. The membership of a project group is ordinarily controlled by the manager of that

project.

3. Interest groups, which are ad-hoc collections of individuals who are interested in sharing information about some subject. The access controls on interest groups are generally arranged so that any individual can add or delete his own R-Name.

The distinction between project and interest groups is not always clear-cut. A good guideline is to assume that any group whose membership is centrally controlled by one or a small number of responsible individuals is a project group; all other groups are interest groups. Equivalently, any group whose "Friends" list is non-empty or whose "Owners" list is itself a group is probably an interest group.

With this classification in mind, we now continue with the registry administration policies.

▸ 3. Do not permit interest groups to be used for IFS access control.

That is, an IFS administrator should not specify an interest group as one of the groups usable for access control on the IFS. This should be obvious: a group to which any individual can add his own R-Name is a totally ineffective basis for access control.

▸ 4. On U.S. IFSs, do not permit groups containing non-affiliates or foreign affiliates to be used for IFS access control, except by prior authorization. Exceptions to this rule must be approved by the International Deputy (see section 5.1).

This is required to ensure that such individuals cannot gain access to information in violation of the security guidelines. Exceptions to this policy for specific projects and individuals may be authorized by the International Deputy on a case-by-case basis.

3.3. International information transfer

Under U.S. Government regulations, most transfers of information to foreign nationals (whether Xerox affiliates or non-affiliates) are required to be logged. Since no automatic logging mechanisms presently exist in the Xerox Research Internet, such transfers cannot be permitted via the Grapevine and IFS access controls. Instead, some more centralized and restricted procedures are required.

Note that this requirement applies to technical and business documents, but not to casual interpersonal correspondence conducted via electronic mail. The latter corresponds to first-class letters in the postal system, which are also not subject to any regulations. Of course, the distinction between a "document" and a "letter" is not clear-cut; the sender must exercise some judgement in determining whether it is appropriate to send a particular piece of information by electronic mail.

To ensure the necessary control, it is required that information transfers from U.S. organizations to foreign affiliates be coordinated by the Export Control Coordinator for the originator's organization. This will also ensure that the required records are maintained.

The mechanism for transfer is that dedicated IFS directories be established that are accessible for writing by the Export Control Coordinator and for retrieving by persons in the registries established for foreign affiliates (or preferably by the Technical Information Center of the foreign affiliate, which will then distribute the documents further as appropriate). The Export Control Coordinator will determine whether the file is cleared for transmission to the foreign affiliates, move the file to the dedicated directories, and log the transfer.

4. Commonly-occurring problems

In this section we consider various situations that have arisen in the past and that have sometimes been handled in ways that violate the above policies. Historically such practices developed because the authentication and access control mechanisms were inadequate. Now that the Grapevine facilities are available, these practices are no longer necessary and should be abolished.

4.1. Communal R-Names

Sometimes R-Names are assigned for communal use by multiple people. Strictly speaking, this does not result in violation of any information security requirements so long as all users of the communal R-Name have exactly the same status (i.e., are members of the same organization and projects).

But there are many disadvantages to this. When no single user is personally accountable for use of the R-Name, it's hard to detect and control unauthorized use. When one of the users transfers out of the project or leaves Xerox, maintaining the R-Name's integrity requires changing its password, which inconveniences all the other users (with the consequence that the password usually doesn't get changed). It's hard enough for administrators to keep track of organization and project membership without having to worry about informal "groups" of users who share a single R-Name.

For these reasons, the policy of assigning an individual R-Name to each user should be rigidly adhered to. If a person has legitimate reason to use the Xerox Research Internet, then that person should be registered—without exception.

4.2. Institutional R-Names

A related case is the one in which an R-Name represents not a particular person but some function, organization, or service; the R-Name is assigned for the convenience of people attempting to access the entity it represents, so they need not remember the R-Names of the individuals who actually embody that entity. Examples of such R-Names are LaurelSupport.PA and NetSupport.Wbst.

In this situation it is usually most appropriate for the R-Name to identify a *group* instead of an *individual.* The members of the group are simply the R-Names of the people representing the named entity.

Sometimes institutional R-Names have been registered simply so that messages can be sent on behalf of those institutions. However, this is not necessary, since Laurel permits the originator to specify the "From" and "Reply-to" fields of a message explicitly; for example, a member of the LaurelSupport group can compose a message that says it is "From: LaurelSupport.PA". In this case, Laurel adds a "Sender" field to identify the actual originator of the message.

4.3. Delegated access

Sometimes a user will desire to delegate access or authority to another person. For example, a user may want his secretary to read his mail while he is on vacation, or may want to make some private file available to a second person. Users sometimes give out their passwords to others under such circumstances.

This is *never* appropriate, and *always* violates the information security requirements. It should be impressed upon users that they are to keep passwords secret and never divulge them for any reason whatever.

An individual's incoming mail may be temporarily diverted simply by establishing a "forwarding" entry for that individual in Grapevine.

In general, users should be encouraged to work within the system when transferring information to others. If some individual does not have access to certain information, it is usually because the individual is not a member of some group that he should be in, or because the information's access control list has been set incorrectly. Smooth information transfer requires that people understand how the protection system works and how to use it.

4.4. Automatic access

A number of programs have been developed that access IFSs using a compiled-in R-Name and password instead of the credentials of the human user running the program; examples of such R-Names are Guest, ARUser, Librarian, and Smalltalk-User. The existence of such R-Names constitutes a security loophole, for reasons already presented, and they must be abolished.

Most uses of compiled-in credentials exist for no better reason than that the implementors of the software consider it too inconvenient to obtain the human user's credentials. Users are justifiably annoyed when they must log in repeatedly because the software they are running forgets their credentials in mid-session (e.g., because a new Pilot volume is booted); alleviating this annoyance is a likely reason for the widespread use of compiled-in credentials. But the resulting adverse impact on information security makes this practice no longer tolerable.

In a relatively small number of cases, such "automatic" access really is required because human interaction is impossible for some reason. For such applications, it is possible to establish individual R-Names representing non-human entities; but these R-Names must be in *special registries* that do not also include human individuals. That is, it is unacceptable for such R-Names to be in registries such as PA, ES, and Wbst. (For example, the existing RS and MS registries contain R-Names representing the individual Grapevine registration and mail servers.)

Access by such non-human individuals to information must then be precisely controlled by proper definition of groups and use of access control lists. Such individuals are not in any IFS's "World" group since their registry is not among the ones included in USRegistries↑.internet; thus their credentials cannot be used to subvert the protections of proprietary information.

System implementors who elect to adopt this strategy should be aware that all information obtainable by such "automatic" means may also be accessed by *anyone* able to obtain a copy of the software (or discover the R-Name and password by other means), regardless of who or where he is.

To summarize: a program accessing information on behalf of some human user must obtain and present that user's credentials—without exception. Only when human interaction is *impossible* (e.g., in servers that run unattended) should credentials be compiled into programs or otherwise stored for later automatic use; but such R-Names must be in special registries that are not included in USRegistries↑.internet.

5. Additional information

5.1. People

Horace Becker (8*222*2163) is the International Deputy for Reprographics and Jerry Elkind (8*923*4610) is the International Deputy for Non-Reprographics. Each organization has its own Export Control Coordinator.

5.2. Documentation

More detailed information about electronic information security and proper use of the authentication and access control facilities is available from several sources:

1. "Electronic information security guidelines", in the October 1981 issue of the Xerox Research Internet Newsletter.

2. "IFS meets Grapevine", in the March 1982 issue of the Xerox Research Internet Newsletter.

3. "How to use IFS", filed as <IFS>HowToUse.press on many file servers.

4. "Maintain reference manual", filed as <Laurel>Maintain.press on many file servers.

ic: *H. Becker*
J. Schweitzer
Export Control Coordinators

```
[PHOTO:  Recording initiated  Mon 10-Jun-85 11:31AM]

 End of COMAND.CMD.2
@telNET.EXE.939
TELNET>ifs
 Trying... Open

Stanford-CSD IFS 1.38L, Executive of October 13, 1983; 1 user out of 9.
@statistics (for) memory

0 smallZone overflows, 0 bigZone overflows
Overflow pages: present = 0, maximum = 0
Net blocks allocated minus blocks freed: 247
33 total PBIs; overflow PBIs: present = 0, maximum = 0
33 total VMem buffers, 0 buffer shortages
VMem    Overlay    VFile       DD     Leaf
Reads       116     2196      139        7
Writes        0      687     2827       15
189 XM pages; 98 overlay reads from XM, 18 from disk
Leaf VPBI reads: 0 actual / 7 total
Leaf VPBI writes: 5 actual / 14 total
VPBI reaper scans: 1 ancient / 11 normal / 25 total

Main memory errors:
Card  Chip  Errors
  2    30       2
  4    49      12
  2    26       1
@quit [Confirm] yes.

Connection closed by foreign host
TELNET>qUIT (OUT OF TELNET)
@pop
```

forcing ifs to act as a time server:

get into swat (cntl-shift-swat) while ifs is running.
(ie: displaying "IFS")

~~→ifs/t~~
~~(falls into swat)~~

^Y IFS (gets IFS.SYMS)
@ ts+13 ^O (displays "addvess : 1")
O ccr> (turn on flag)
^P (proceed)