XEROX
BUSINESS SYSTEMS
*System Development Division*

To: Distribution

Date: June 15, 1978

From: W. C. Lynch

Location: Palo Alto

Subject: Scavenger, Copydisk, InitializeDisk

Organization: SDD/SD/SSW/Pilot

Filed on: [Iris]<Lynch>Scavenger.Memo

| | | | | | | |
|---|---|---|---|---|---|---|
| Copies: | Archives | Belleville | Bergsteinsson | Bewley | Clark | DeSantis |
| | Harslem | Heinrich | Irby | Kennedy | LeCesne | Liddle |
| | Lynch | Mendelson | Metcalfe | Reilly, D. | Reily, J. | Schwartz |
| | Sonderegger | Szelong | Thacker | Townsend | Wallace | Weaver |
| | Wick | Wickham | White | | | |
| Copies: | Lauer | Redell | McJones | Purcell | Dalal | Hankins |
| | Jarvis | Horsley | Murray | Kierr | Sandman | Johnsson |
| | Ogus | Garner | Bowering | | | |

## Introduction

The situation regarding a Scavenger, Copydisk, and InitializeDisk for Oak has been unclear. This memo is intended to report the current status of those items and to propose a resolution of the items.

## Definitions

InitializeDisk (Pilot) - A program which takes a virgin disk from the manufacturer and formats it into a Valid Pilot Disk containing no Pilot files.

Valid Pilot Disk - a disk pack which has been formatted with the proper header, label, and data blocks. In addition, bad spots have been identified, marked and removed from service.

Valid Pilot Volume - A Valid Pilot Disk which additionally contains a proper Pilot Volume as described in [Iris]<Purcell>PilotVolumeFormat.memo (attached).

Copydisk (Alto) - An existing Alto program which makes a bit-for-bit copy of one disk pack on another.

Copydisk (Pilot) - A Pilot application client which makes copies of all of the files on the source Pilot disk upon the target Pilot disk. This differs from a bit-for-bit copy in that multiple FIDs for mutable Pilot files must be avoided.

Movedisk (Pilot) - A Pilot application client program which makes a bit-for-bit copy of one Pilot disk pack on another. It differs from the Alto Copydisk in that the source pack must be erased or otherwise made permenantly unavailable so as to avoid duplicate FIDs for the same mutable Pilot file. This should be used only when a pack contains only immutable

files or when the physical integrity of the pack itself is suspect (and the user wishes to discard the physical pack but not the information on it)

Scavenger (Alto) - An existing, relatively ill defined (see below) Alto program which takes a not-too-badly smashed Alto disk and makes it acceptable to a certain set of Alto subsystems.

Scavenger (Pilot File) - A Pilot client program which takes Valid Pilot Disk and leaves it containing an undamaged Valid Pilot Volume.

Scavenger (Pilot Disk) - A program which produces a Valid Pilot Disk from a damaged Pilot disk. It identifies and records bad spots and attempts to relocate the overlaid information to other places on the disk.

Scavenger (Star) - A Pilot client program which repairs damaged Pilot client objects which are stored in Pilot files.

## Status

1) Bob Bowering has been in the hospital and unavailable for consultation. His continued availability is uncertain.

2) Steve Purcell has written a memo (attached) capturing the information required to deal with the Pilot disk and required to construct a Pilot file scavenger for it. (There are other kinds of scavengers, e.g. a Star document scavenger, which will not be discussed here. See above.)

3) I have talked with Jim Morris (the resident Alto scavenger wizard) at some length about the status and functions of the Alto scavenger.

4) I have determined that the current Alto Copydisk will function very well as the putative Pilot MoveDisk.

5) Steve Purcell has already constructed an InitializeDisk routine which, with minor packaging, can be delivered with Oak to serve to initialize Pilot file disks. It does not deal with bad spots.

## Alto Scavenger

I wish to record here some facts about the Alto Scavenger which I gleaned from my conversation with Jim Morris. As the author of the Alto Scavenger he has been subjected to a wide variety of house calls for problems of one kind or another.

1) Jim confirmed our impression that far-and-away the most important thing is the reconstruction of smached and invalid directories and allocation tables.

2) There was never a good definition of what the Scavenger would do, of where its duties would leave off and a subsystems duties would begin. Ex post facto negotions between the Scavenger and the major subsystems have left a lot of anomalies, rough edges, and an endless wish list.

3) There are specific error modes which have accounted for the bulk of the problems. These are:

a) Bad Spots - The current scavenger cannot deal with bad spots. Mike Overton is slowly accumulating unusable disks that have bad spots on them. As a result I am placing low priority on dealing with bad spot problems in Oak.

b) Disk Alignment - This is now less visable as many more people have their own Altos and packs are shifted less frequently. Jim had a scheme worked out (the details of which he could not recall on the spot) which would detect incipient misalignment before it became a real problem. It required co-operation from the drive manufacturer.

c) Power Supply run-away - Many problems could be attributed to misbehavior on the part of the power supplies, causing bad writing on the disk. (I would catagorize our problems with IFS during power failures here)

d) Processor Overload - The Alto has had problems with the microcode tasks not reacting within real time constraints under unusual circumstances. (There is an infamous bug which caused every sixth FTP page to be badly writen due to a combination a microcode tasks collectively taking too much time.)

e) The generation of UIDs is poorly done (it's more like a bug) causing more than one file to have that same UID. This complicates life for the Alto Scavenger. Jim agrees that Pilot has that problem designed out. The problem has never been corrected on the Alto simply because the system is not being maintained (the specific problem seems easy to fix).

## Proposal

1) Steve Purcell is to be directed to specify, document, package, and deliver to Oak alpha test an InitializeDisk. In Oak it will not deal with bad spots.

2) The current Alto Copydisk be used as the Pilot Movedisk in Oak. This will be run on an Alto. Removing the old pack from circulation will be accomplished by operational procedures. No Pilot Copydisk will be provided with Oak.

3) That Steve Purcell be directed to specify, document, and deliver a Pilot File Scavenger which is restricted to reconstruction of the vfm and vam. Neither a Pilot Disk Scavenger nor a Star Scavenger will be delivered with Oak.

To Distribution Date June 13, 1978

From Stephen Purcell Location Palo Alto

Subject Pilot Volume Format Organization SDD/SD

XEROX

Filed on: [Iris]<Purcell>PilotVolumeFormat.memo

This memo describes some aspects of Pilot Volumes. Since the design of Pilot and its volumes is still in flux, only a snapshot of the current design and implementation can be provided. There is no guarantee that what follows will not change. The general structure is probably correct and will remain, but many details will change.

Pilot stores files in volumes which are physical disks with data conforming to certain constraints. Pilot assumes the storage to consist of pages randomly accessed by *volume-page-numbers* which range continuously from zero to the size of the volume in pages. Bad Pages will be hidden by the disk driver. A single disk Diablo 31 will have volume pages in the range [0..4872). As far as Pilot's volume manager is concerned, a page is an 8 word label and 256 words of data. This memo ignores additional page fields such as the 2 word address header used by the device drivers/controller.

Pilot partitions pages into a number of files, each with a unique identifier (UniversalID) and a type. (The generation and registration of types is still fuzzy, but many files can have the same type). (The file properties of immutable and temporary may be viewed as contained in the type, although they are actually independent fields.) Pilot uses four types for *system* or *volume* files, present on every volume. Each page in a volume belongs to exactly one file and has a label with the file ID, the file type and the *file-page-number*. A client file has pages with consecutive file-page-numbers from zero, while a system file is numbered by volume-page-numbers, which are not necessarily consecutive. Page labels and page data are stored together for safety. For efficiency and redundancy, label information is also stored in the system files, which can be entirely discarded and reconstructed from labels if damaged. The four system file types are *root, vam, vfm* and *free*. The root file is exactly one page with a constant location on the volume, containing IDs (and sometimes page numbers) of the other system files and of one client root file. The vam (volume allocation map) is a bit map telling which pages are free. The vfm (volume file map) is a B-tree which maps (file ID, file-page-number) keys into volume-page-numbers for all client files. The free file has blank pages scattered over the volume that are not in use either by clients or by Pilot. Bad pages can be thought of as free pages but the Pilot volume manager is not aware of them, since the disk driver ensures that there is a good page for every volume-page-number.

The root file is created and accessed by

```
LogicalVolume.RootAccess: PROCEDURE[volume: Volume.ID, proc: PROCEDURE[volume:
LogicalVolume.Handle]];

LogicalVolume.Handle: TYPE = POINTER TO LogicalVolume.Descriptor;

LogicalVolume.Descriptor: TYPE = RECORD[
    version: CARDINAL,
    ...
```

```
    volumeSize: Volume.PageCount,
    vID: Volume.ID,
    vam: File.ID,
    vfm: File.ID,
    free: File.ID,
    vam: File.ID
    ...];

LogicalVolume.nullDescriptor: LogicalVolume.Descriptor= ... ;
```

VolumeRootAccess is a stylized way to read and lock the root page, access and modify it by a client procedure **proc**, and then write it back to the volume and unlock it. The proc can use the handle (a pointer) to read and write the root page which will then be written back to the volume.

The vam(volume allocation map, a bit map) is created and accessed by

```
VolAllocMap.Init: PROCEDURE[volume: Volume.ID];

VolAllocMap.GetBusy: PROCEDURE[volumePage: LogicalVolume.PageNumber] RETURNS
[busy: BOOLEAN];

VolAllocMap.SetFree: PROCEDURE[volumePage: LogicalVolume.PageNumber];

VolAllocMap.GetSetBusy: PROCEDURE[volumePage: LogicalVolume.PageNumber] RETURNS
[busy: BOOLEAN];
    --set a page to busy and return its previous state

VolAllocMap.GrabFirstFree: PROCEDURE[volumePage: LogicalVolume.PageNumber] RETURNS
[LogicalVolume.PageNumber];
    --find first free page and set busy (may signal Volume.InsufficientSpace)
```

The vfm(volume file map, a B-tree) is created and accessed by

```
VolFileMap.Init: PROCEDURE[volume: Volume.ID];

VolFileMap.GetPageGroup: PROCEDURE[file: File.ID, filePage: File.PageNumber] RETURNS
[FileInternal.PageGroup];

VolFileMap.GetNext: PROCEDURE[file: File.ID, filePage: File.PageNumber] RETURNS
[nextFile: File.ID, nextFilePage: File.PageNumber];
    --starting and ending with null, enumerates the page group boundaries

VolFileMap.InsertPageGroup: PROCEDURE[file: File.ID, group: FileInternal.PageGroup];

VolFileMap.DeletePageGroup: PROCEDURE[file: File.ID, group: FileInternal.PageGroup];
```

The vfm maps keys (file ID, file-page-number) into volume pages, and is abstractly a collection of entries (file ID, file-page-number, volume-page-number). The procedures for accessing it use *page groups* to encode runs of entries with file-page-numbers in a closed-open interval: [..). The *null* volume page resulting from initialization or deletion signifies the absence of a file or a page of a file. Entries have unique keys, and insertions overwrite existing entries. Therefore pages with duplicate ID and page number cannot be pointed to by the map. A scavenger would have to deal with such (illegal) page pairs before updating the vfm. The vfm does not depend on consecutive file-page-numbers so that as it is being reconstructed, say by the scavenger, it can contain fragments of files. Client files with gaps, however, are not permitted in a legal Pilot volume. Insertions are most efficient when clustered by key (ordered by ID, page number). GetNext is used as an enumerater. All client file pages can be located on the volume in random access fashion by use of the vfm.