Inter-Office Memorandum

To Bill Lynch

Date October 25, 1977

From Peter Bishop

Location Palo Alto

Subject Comments on Pilot File System

Organization SDD

XEROX

Filed on: <Bishop>filerec.memo

The following are a set of thoughts and recommendations on the Pilot File System. They range from specific proposals to continuing discussions on various points.

1. Making Files Objects

I think that there will be several pieces of software on OIS that will need to treat files as objects. In particular, the CopyDisk operation will define a Copy operation on every type of file that must correct any fileIDs kept within the file. Plans seem to be solidifying to make documents reside in a single file. Documents will be treated as objects by the desktop. A sophisticated scavenger needs to treat files as objects so that when a single page is lost, the rest of the information in the file can be saved.

If files are to be treated as objects, then it would be valuable to have one of the attributes of a file be its *type*. This would be an 8 or 16 bit number that would be carefully allocated to various subsystems so that no two subsystems would have the same type of file unless the software to manipulate the file was interchangeable. In order for the scavenger to be able to use this type, however, it is necessary for the type of the file to be placed in the page header in addition to the fileID and the page number within the file. This protects against loss of the page that contains the attributes of a file.

2. Owner Counts

The original Concepts and Facilities suggested that each file should have a reference count. I do not believe that files should have reference counts, but I think that it would be valuable for files to have *owner* counts. Directories *own* the files contained in the directory. Similarly, the desktop *owns* any file that is currently sitting on the desktop in iconic form or as an enlarged window. It should be possible for the same file to be in a directory and on the desktop at the same time. When a directory removes an entry from itself, ordinarily it would delete the corresponding file. I am suggesting that the directory should merely decrement the *owner* count. When the *owner* count is decremented to zero, the file is deleted. Each copy of an immutable file has its own copy of the *owner* count. Thus it is necessary to specify the location of the file whose *owner* count is to be decremented.

Another use of the owner count is for identifying temporary files. Files are initially created with an owner count of zero. Whenever the system is brought up, all files with owner counts of zero are immediately deleted. When a program constructs a permanent file, it creates a file with a zero owner count, initializes the file, and then increments its owner count to one. At this point, the fileID of the file has been stored in some sort of directory that resides on disk so that even if the system crashes, the system will know that it is responsible for this file without performing a garbage collection. This feature can drastically reduce the need for garbage collection during crash recovery.

3. File Location Handles

I noted that Hugh suggested that we scrap Volume Capabilities in favor of Volume IDs. I liked Volume Capabilities for the following reasons.

There are basically two kinds of software that use a file: the programs that do not care where the file is stored and those that do. Most of the software that manipulates the contents of a file is not concerned with the location of the file, and so it uses a File Capability to manipulate the file. A directory, however, such as a file cabinet, is responsible for the storage used by the file and so is very concerned about the location of the file. A directory, therefore, does not really want merely a File Capability for the file, it would prefer to have a File Location Capability that specifies not only the FileID but also the Volume ID on which the file resides (a FileID does not uniquely determine a particular *copy* of a file). Directories need to deal with the Volume itself, however, (Creating new files on the volume.) so it is necessary to have some sort of Volume Handle anyway.

I thought that the purpose of the Volume Capability was to provide a Volume Handle that, when used in conjunction with a File Capability formed a File Location Capability. It seems clear to me that the delete operation should be specified in the File Location Capability instead of in the File Capability (the *owner* of a file has a File Location Capability for the file with the delete permission turned on). That is, only a piece of software that is responsible for storage should be deleting files and all software that is responsible for storage must at least know what volume the file is on. Thus it seemed clear to me that the File Capability should not contain a delete permission, and that the delete permission in the Volume Capability was conceptually the delete permission that properly belongs in the File Location Capability. I did not find it particularly disturbing that a File Location Capability never really existed, it merely consisted of passing both a File Capability and a Volume Capability to Pilot. This reflects the fact that a single directory will probably be responsible for files on a single volume, and so the directory only needs to store one copy of the Volume Capability instead of including it in every File Location Capability. Once we get to card catalogs that are concerned about the locations of files on many volumes, then File Capabilities will naturally be stored next to Volume Capabilities.

This analysis also suggests that a Volume Capability should be specified even when deleting mutable files even though this is not strictly necessary.

4. Clearinghouses and Pilot

Hugh's memo suggests that if Pilot cannot find a file, that there is no point in allowing the computation to proceed. This is not true, however, if a higher level piece of software exists that is able to bring a copy of the file that Pilot tried to find onto the local system. It seems to me that the problem of finding files is a difficult problem on distributed systems and so it is quite possible that a piece of software that is able to run on top of Pilot may be able to find and obtain a file that Pilot was unable to find. On the other hand, we will not need this feature immediately. As long as we realize that it may be necessary to put hooks into Pilot in the future to allow a clearinghouse to look for a file, I do not object to removing mention of the clearinghouse from Pilot. I think that it will probably be easy to add this feature to Pilot in the future. Since no one will depend upon the feature, it need not be described until someone wants to use it.

5. FileIDs

Hugh's recent memo is now the second memo that he has prepared that talked in vague terms about problems with FileIDs without proposing any solutions. The memos that other people have prepared, however, have dealt with specific proposals. I do not think that it is fair for Hugh to write into the Functional Specs his first draft of a proposal for FileIDs when other people have shown by example how difficult it is to arrive at a satisfactory solution to this problem (I don't know that Hugh was going to write his proposal into the

Function Spec, but why hasn't he given a written proposal that other people can criticize?). It is not fair for Hugh to be able to shoot down concrete and well-thought-out proposals with vague problems when it is not clear that he can solve as many problems as do the solutions that he is criticizing.

Size: Hugh claims that my recent proposal for creating IDs assumed a 20-bit processor serial number. In fact, I did not suggest that the IDs allocated to a processor have any relationship to the processor serial number. I did suggest that we be able to perform a total of 2^{24} allocations of fileIDs to processors.

Generation: A little while ago, Paul McJones suggested to me that if we have rigid disk on the system, it might be reliable enough to store the ID allocator that exists within the processor on the rigid disk. It would be cheap to store three or four copies of the allocator on the disk to protect against disk crashes. Thus we may not need any non-volatile memory in the D0. This technique could at least be used as an interim measure since even if it wastes too many IDs we could afford to use up 10% of the total FileID space on early versions of the system.

Maintenance: In my memo on 64-bit fileIDs I suggested that there is a significant cost savings in using small FileIDs. Any discussion of maintenance costs must take into account these cost savings. I suggested that the maintenance cost could be made arbitrarily small by making the need for it very infrequent. Hugh argues that if the maintenance operation is very infrequent then the maintenance people will forget how to service the problem. The problem of diagnosing the problem can be solved by having the system explicitly tell the maintenance man what the problem is. The problem of forgetting how to perform the maintenance can be handled by having the system tell the maintenance man where to find the directions for performing the maintenance. There is no question that the fact that this problem is very rare will increase the maintenance cost of the operation, but I doubt that the cost will more than double, which can be offset by reducing the frequency of the operation by a factor of two. Hugh further suggests that the return for solving the problem will not be significant. I disagree. For one thing, there is a savings in storage for all of our customers. Second, Hugh's comment makes me wonder if it will be possible for Hugh's scheme to overflow. If so, then if we do not perform an equivalent maintenance operation then the customer will either start reusing fileIDs which will eventually conflict with one of the user's own files (making Xerox look very foolish) or the customer will be forced to buy another system, which will enrage the customer. These possibilities all result in extreme customer dissatisfaction with our product, which has a very high cost indeed.

Binding: I find it difficult to understand why Hugh is so afraid of the possibility of Universal File IDs and why he insists on conducting his education on this point in print. Isn't there a more amicable way to conduct our affairs?

Hugh raises the question of how one file will refer to another file. I think that whenever a file wants to refer to another specific file, the first file will contain not only a File Capability but also a character string that is meaningful to the people who read it and which identifies to those people the file that is specified by the File Capability. In other words, the File Capability is the system representation for that character string. Implicit in this scheme is the idea that a particular file, A, should be bound to another specific file, B, if the name of B is present in A. In my memo on the theory of names, I tried to suggest that the File Capability that is stored in A should have the exact same meaning as the character string name that it is associated with. This means that it never becomes necessary to modify the File Capability stored in A unless it is also necessary to change the character string name. All problems of binding the File Capability for B to the appropriate file should be considered to be B's problem (or at least the problem of the person in charge of defining the value of the File Capability for B that was stored in A).

What is the character string that is stored next to the File Capability used for? Most of the time it is just used to display to the user which file is meant, since File Capabilities are never displayed to the user. If, however, the low level software cannot find the file given its File Capability, then this character string becomes useful in allowing the user to find the file. The user must at this point perform an information retrieval task to find the file. The character string name assists him in performing the information retrieval task, partially because the name contains some information about the file, but also because the name identifies the file to the user who may know some additional information about the file. The information retrieval task consists of going to libraries of various types and trying to find the file in these libraries. At this point, the presence of the File Capability greatly assists the information retrieval task, since if the library has that particular file, then we can be sure that the right file has been obtained. It may be, however, that the exact version of the file that is needed is no longer available, in which case the File Capability may not be of assistance, but if the information retrieval task is completed by the user then it may find a similar version of the same file. The user is in a position to determine whether this version is acceptable since the character string name of the file identified the file to the user. The justification for having these two names, the character string name and the File Capability is that the character string name can only be understood if one understands the context in which it is being used. Since the system is not very good at dealing with contexts in which words are used, while people are very good at it, it seems to be advantageous to convert a word to a Universal Name Space while we are still in the appropriate context. It is then easy to communicate information between two different contexts. If the English discription is no longer adequate, then the user can merely specify a different set of words that, in the new context, has the same meaning as the old set of words had in the old context.

Thus we see Hugh's comments on binding in a new light. The only reason for looking up a character string in a "directory" is when the user is really performing an information retrieval task. Such tasks are most appropriate for large data bases, such as the card catalog of a library. For such a data base, it would be very helpful if the File Capabilities were within a Universal File ID space for two reasons: 1) they may well point to files on different systems and 2) they allow the File Capability that the user has to be used to reduce the work for the user involved in the information retrieval task (by searching the entire Card Catalog for that File Capability before really beginning the information retrieval task).

The problems that Hugh predicts for handling different versions of a file is a common problem when constructing object code, but it is a less serious problem in the world of documents. Cross-references from one document to another should never be changed, just as typos in an immutable document should not be corrected. Most of the time when one document refers to another document, this reference should not be changed when a new version of the document is created because the part of the document that was changed is likely to be the part that was discussed in the first document. In those few cases when a document should refer to a mutable object that may change state, the File Capability should identify the exact mutable object that the document wishes to refer to. The problems of binding this File Capability to the appropriate file can then be solved without needing to modify an immutable document.

Both the problems of constructing data bases capable of sophisticated information retrieval and of binding File Capabilities to other files are problems that have been recognized, but these problems have been seen as being problems that need not be solved immediately. Both of these problems probably cannot be solved as nicely as we would like because we notice that current filing systems and offices have these problems and have not found nice solutions to these problems. This suggests to me that it is not necessary to arrive at total solutions, it is sufficient to arrive at solutions that are as good as the solutions that current filing systems and offices

already have. It should not be difficult to provide such limited solutions.

Recommendation: Proceed with Universal FileIDs because there is no pit to fall into.

c: Liddle
Shultz
Redell
McJones
Lauer
Horsley
Gifford
Moore