XEROX
BUSINESS SYSTEMS
Systems Development Department

To: Distribution
Date: November 3, 1978
From: H. C. Lauer
Org: SDD/SD
Subject: **Some Thoughts on a Pilot Scavenger**
Filed: [Iris] < Lauer > Scavenger.memo

In this memo are presented some thoughts on the problem of scavenging Pilot volumes containing interesting client data structures. These were discussed briefly at a meeting between Abraham, Bishop, Lauer, and Reiley on November 3. The ideas expressed in that meeting and in this memo are to be considered 'unbaked.' Your comments are invited.

General Framework

The current situation is that a crude scavenger is buried in Pilot and is invoked whenever a (potentially) malformed Pilot volume is detected at startup time. This is only a temporary arrangement, and there are a lot of possibilities for a better scheme to support

a. the development of client software, and
b. products in the field.

Unfortunately, it seems likely (but it is not necessary) that different scavengers will be needed for these two different requirements. I would like to propose a framework in which both kinds of scavengers can be implemented.

I propose that all Pilot scavenging and at least some client scavenging be a stand-alone activity and should not be embedded in Pilot. The scavenger either could be called (i.e., booted) automatically when Pilot detects a problem at, say, boot time, or it could be called in response to some client or user action. The scavenger would do two things. First, it would put the Pilot files and volume back together, as it does now. Among other things, it would mend code and image files as necessary. Second, it would provide an interface for client scavengers to mend client data and directory structures. At the very minimum, this interface must enumerate the files on a volume, providing their File ID's, attributes, and an indication of which pages were found to be smashed or missing. It is probably desirable that Pilot provides this enumeration in some structured way under client control, such as, for example, an enumeration of files of only a particular subrange of **File.Type**. When the Pilot scavenge is completed, one or more client scavengers are invoked (in a proper order, which is probably established at the time the system element is installed).

There are two basic strategies by which the Pilot scavenger can feed information to client scavengers. One method is the procedural interface. The Pilot scavenger and the client scavengers communicate by calling procedures back and forth. Files in the range to be enumerated are passed as parameters, in the style of a *generator* in Mesa, in an array of descriptors, or by some other convention. The second method is for the Pilot scavenger to list the client files, along with useful information, on a log file left in an agreed place. The client then performs much of its 'scavenging' as part of its ordinary initialization at boot time. In this case, the client scavenger does not communciate directly with the Pilot scavenger; however, some means of parameterizing the Pilot

scavenger may be necessary in order to enumerate the correct set of files. (This latter approach might be taken, say, by a transaction data management system which needs to re-execute all outstanding intentions lists. The scavenger needs only to find the intact lists and leave pointers to them. The data management initialization code then executes each list, using the full power of Pilot and other common software.)

The general solution to this problem is to provide the procedural interface between Pilot and client scavengers but to allow the client scavenger to default to a very small nub if desired. For example, this nub might only write (or cause the Pilot scavenger to write) a log file in some structured way.

Some trade-offs

There are a number of issues which need to be resolved and which involve a trade-off of conflicting constraints or objectives. Two of these are discussed below. There are, no doubt, others equally interesting.

First, for the client scavenger, there is the trade-off between having available the full power of Pilot for support as opposed to operating in a very safe, simple environment. The full power means that the client scavenger is almost like an ordinary Pilot client. It could create files, map spaces, access the Xerox Wire, talk to the user, etc. (Note, however, that the current Pilot **File** interface does not provide a rich enough set of functions to allow a client scavenger to be built directly on top of Pilot. Among other things, there is no function to enumerate the files on a volume.) The disadvantage is that scavenging would be taking place in the very environment in which the breakage or crash occurred.

By contrast, a separate, stand-alone scavenger would operate in a very limited world with limited facilities. Presumably, this would be more reliable, and perhaps more efficient. Scavenger code and data structures, for example, would be resident in real memory and not subject to paging from the very volume being scavenged.

Another trade-off is whether the client scavenger gets access to the files its is scavenging through the conventional Pilot file machinery or by some other means. The 'full' Pilot machinery would, of course, be available in a scavenger running on top of Pilot. It might also be included or copied in a stand-alone scavenger. Alternatively, a stand-alone scavenger might provide some other means for accessing the files, perhaps part of a scan across the disk. This has important efficiency considerations. If, for example, a Star scavenger has to touch every Star file as part of the process of putting its directory together again, accessing them via the Pilot B-tree and file machinery could cause a lot of arm wagging and take many minutes in scavenging time. A stand-alone scavenger could provide an interface to do the same thing in a single scan at disk speed.

Note that a scavenger is likely to want to manage real (as opposed to virtual) memory more carefully that an ordinary Pilot client would. This is another facility that a stand-alone scavenger with a client interface should provide.

A name

I got very tired typing the phrase "a Pilot scavenger." Therefore, I propose that such a thing be given a name to be used in future discussions; my choice *Sumac.*

c: W. Lynch, T. Linden, P. Bishop, R. Moore, S. Abraham, J. Reiley, J. Frandeen, D. Redell, S. Purcell, P. McJones, T. Horsley; Other interested parties