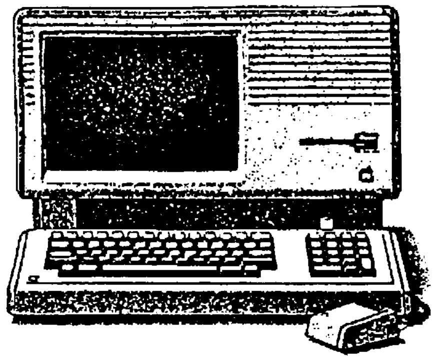
Apple Lisa Information



FILE NAME
Integrating with the Filer

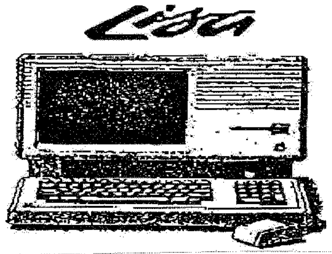
DISK #

COMMENTS
23 Sep 1981

3 pages

David T. Craig
736 Edgewater, Wichita, Kansas 67230
(316) 733-0914

Lisa



TO: Lisa Applications

SUBJECT: Integrating with the Filer

FROM: Frank Ludolph

DATE: September 23, 1981

Re: Preliminary Filer-Applications Interface, May 13
Results of the Filer-Applications Interface Meeting, May 20

filed on: T:IntFiler

This memo is offered in the spirit of 'Let's learn what it takes to integrate' month and to aid the various projects with their scheduling. While there has been some discussion of Filer-application interaction in the past, changes and addenda make it appropriate to recapitulate the things that a Lisa application must do with respect to the Filer. The activities can be broken into 5 phases: initialization, document open, document close, termination, and document copy. Window manager terms are used where possible.

INITIALIZATION

The Filer will start up an application when the user unfiles a document that requires the application. The application should immediately declare a terminate exception handler via O/S DECLARE_EXECP_HDL (the content of this handler is discussed under 'TERMINATION' below). The application should then go through whatever general one-time (document non-specific) initialization is required and call the window manager Get-Next-Event. While the application is waiting the Filer will tell the window manager to create a folder and associate it with the applications process id.

OPEN DOCUMENT

A Lisa document is implemented as a (collection of) diskfile(s); the Filer is unaware of their exact number and names. Each diskfile of the document will have the same diskfile name prefix (the exact string is unimportant to the application). When an application receives an open-document event, tyevDOpen, from the window manager, the pname field in the event descriptor will contain the prefix. The application can then open the document's diskfiles by concatenating that prefix with its own internally supplied strings and calling O/S OPEN.

Two additional 'errors' are being added to the OPEN call: document-not-closed and document-scavenged. The first indicates that the document was not closed by the application when last used, so the application should attempt to verify the consistency of the diskfile's structure. The second indicates that the disk has been scavenged since the diskfile was last closed and that the scavenger found some problem with it (we'll have to wait for a description of the scavenger to find out what sort of things might cause this warning).

If the application discovers that the diskfile(s) is trashed but that at least part of the data can be salvaged, then the application should open a dialog with the user informing him/her of the problem and the proposed action and its consequences and ask for confirmation to proceed. If, however, none of the data can be salvaged, the diskfiles should just be closed without modification and the application should terminate; the Filer will inform the user of the problem. (The reason for this is that it is nearly identical to the case in which the application crashes while opening a document.) See TERMINATE below.

In future releases, the application should check the software version number in the diskfiles and prepare to convert them to the new format if necessary. This means that a version number must be implanted in the diskfiles for the first release. No standard method has yet been defined.

The application should now display the document's content through the folder created by the Filer. (The folder's obox, soon to be fid, is available in the w/m event descriptor.) Any state information saved should be reinstated (see 'CLOSE' below).

A second (third, ...) open-document event will be sent to the application when another document of the same type is unfiled. If an application can handle multiple documents, the open should proceed as described. However, if the application cannot handle the additional document, it should set the fRefused field in the w/m event descriptor to TRUE and then Get-Next-Event. The Filer will start up another process and pass the document to it.

CLOSE DOCUMENT

The application should only close a document in response to a w/m close-document event, tyevDClose. (Exception: see 'TERMINATE' below). There are two kinds of close, the kind being indicated by the fStdClose field in the w/m event descriptor: normal (fStdClose = TRUE) and suspend (FALSE).

If the close is normal, the user is removing the document from the desktop and refiling it. All edits should be made permanent and the diskfiles closed. Whether the current scroll position and other state information is to be saved is currently being discussed.

If the close is suspend, the document is being closed but not being removed from the desktop (in fact the user does not perceive it as being closed). In this case, sufficient state information should be saved to exactly restore the appearance of the folder when it is next opened. In addition the edits should not be made permanent, i.e., they should be undoable via 'Undo All' after the document is next reopened. This is done to support context switching; if the user must temporarily remove a diskette in order to mount another, s/he should not lose anything when the first diskette is reinserted.

After closing a document, the application should prepare to receive another open document event. It is our intent to reuse the process because process start up takes a lot of time.

TERMINATE

An application should only terminate in response to the w/m terminate event, tyevTerminate, or an internal error. Proper termination processing is important in order to prevent system deadlock.

The Filer will signal an application to terminate by sending it a w/m terminate event, tyevTerminate. The simplest thing for the application to do, provided it cleans up things in the terminate exception handler, is to terminate itself via TERMINATE_PROCESS. This will throw it into the exception handler.

As stated under 'INITIALIZATION' above, each application must declare a termination exception handler. This handler will automatically be invoked by the O/S when the process is terminated due to an O/S detected error condition, by itself via TERMINATE_PROCESS or END, or by another process via KILL_PROCESS. The process will die when it returns from the handler or if a second terminate exception is generated for some reason.

THE FIRST THING THAT THIS HANDLER SHOULD DO IS TO CALL THE W/M INTERFACE 'DYING'. Failure to do this will result in a locked-up system which refuses to accept and process user keystrokes. The w/m will activate another process so that there is somebody to run after the terminating exception executes its last instruction.

The application should then try to clean up and die gracefully. Any documents that are still open should be cleaned up and closed as if suspended.

If the process terminates abnormally, the Filer will (in most cases) attempt to restart it and reopen the document(s). The Filer will insure that an abnormal-termination/restart cycle does not loop indefinitely.

COPY DOCUMENT

An application may be requested, via the w/m event tyevDCopy, to make a copy of a document that it has open. The pname field of the event descriptor contains the prefix for the source document and the pnameNew field contains the prefix for the destination document. The application should make a set of diskfiles using the pnameNew prefix and copy the contents of the pname diskfiles to them. When the copy is complete, the pnameNew diskfiles should be closed normally (as opposed to suspended). The source diskfiles should be unaltered.

° The End °