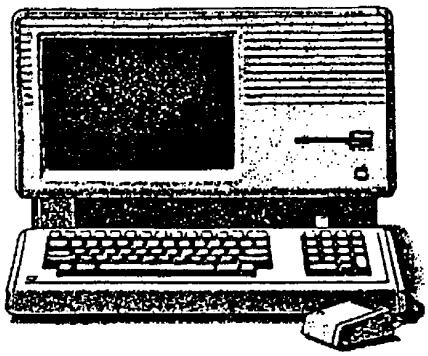
Document# 491

Apple Lisa Information



FILE NAME

Filer-Application Communications

DISK #

COMMENTS

03 January 1983
(Filer ≡ Desktop Manager)

7 pages

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

```
{ FilerComm.text, 3-Jan-83, F.Ludolph } { Copyright 1983, Apple Computer Inc. }

UNIT FilerComm;
INTRINSIC;

INTERFACE

USES  {$U obj:SysCall     } SysCall,
      {$U obj:PSysCall    } PSysCall,
      {$U obj:UnitStd     } UnitStd,
      {$U obj:UnitHz      } UnitHz,
      {$U obj:Storage     } Storage,
      {$U obj:FontMgr     } FontMgr,
      {$U obj:QuickDraw   } QuickDraw,
      {$U obj:WM.Events   } Events,
      {$U obj:WM.folders  } Folders;

{$SETC fcDebug = fDbgOK }
{$SETC fcSymbols = fSymOK }

{ This unit contains the record definition used for Filer-Application
  communications. It is used in both receiving events from and sending
  events to the Filer.

  An application is started by the Filer via the OS call 'Make_Process'.
  The application should execute its initialization code and then
  call GetEvent.  The initilialization code should first call 'OpenWm'
  (to set up the Filer-Application communication channel) and then declare
  a Sys_Terminate exception handler.  If the exception handler cannot be
  declared or if initilization cannot be completed, the application should
  'TellFiler' that 'fcInitFailed' and the reason (see the section on
  unsolicited messages in the table below).  See the Bouncing Balls *
  'Initialize' procedure for an example.

  The Filer sends a FilerEvent to an application. The GetAddParams
  procedure is used to obtain the additional parameters associated with
  this event.  Two parameters are passed: a filerOp that defines the
  operation to be performed, and an optional pathname, fDocName, which is
  used to open, create, and destroy the diskfiles that make up the document.
  An application uses fDocName as a prefix for diskfile pathnames. It consists
  of a disk volume name and the initial characters of a diskfile name.

  There are currently 9 filerOps, those that open a document, those that
  close or copy an open document, one that tells an application to close
  a diskfile, and one that tells the process to terminate.

  Those that open:

     fcNone:    No doc to open. The user pulled a tool rather than a doc.
     fcResume:  Open the doc, or create a new doc if no diskfiles exist, and
                display contents in window. If the doc was suspended, restore
                its state.

  Those that close:

     fcClose:   Update and close doc overwriting the old version.
```

* DTC 4-2001 Bouncing Balls was a program that the Lisa operating system group used to test the OS' responsiveness.

fcCopy: Update doc into new diskfiles and close. The source doc is unchanged and remains open.
fcPut: Update and close doc to new location (fDocName). Destroy the old version.
fcShred: Close the doc as in fcSuspend, if possible, or just close the diskfiles, if possible. Filer will delete them later.
fcSuspend: Close doc, keep edits seperate, save document state.

fcDfClose: Close the diskfile (not document) using the refnum provided else app will be terminated. (User is removing a diskette.)

Terminate:

fcTerminate: Terminate the process and suspend any open docs (actually there shouldn't be any open when this is received).

An fcResume/fcNone is sent when:

1) the user pulls a document onto the desktop (fcResume) or
2) when the user pulls a tool, e.g. the clock or calculator (fcNone).

The window to be used to disply the document is provided by the Filer via eventRecord.who in the open event. The application should never dispose of this window, i.e. call WM.DisposeFolder. NOTE! A reply is no longer expected. The Filer assumes that the document was opened without error. If errors do occur, the application should send an unsolicited docClose to the Filer (see below).

A close type of filerOp is sent if a user puts away a document or its diskette is unmounted. If a document is being put away any edits to the document should be made permanent, however, if the diskette is being unmounted, the document's current state should be saved and the edits maintained seperately.

For first release applications may put away edits or save state as they chose. If time permits put away and save state should be implemented as described above.

A terminate is sent when the application is to terminate, usually because the diskette that holds the application is being unmounted. After calling 'ImDying', if the application still has some open documents, they should be suspended by the application before it terminates.

A NOTE ON 'ImDying': This message should always be sent to the Filer as the first thing done by the application's terminate exception handler, whether in response to a Filer event or unsolicited. If an application terminates before making this call, it is likely the case that all other processes, including the Filer, are suspended and so the system will hang.

When an application has completed processing the Filer initiated event, it sends a response back to the Filer via the 'TellFiler' procedure. (Some events do not require a response.) Appropriate responses for each event are listed below. Note that several operations can be aborted by the user.

In general, the application is responsible for informing the user of any difficulties via the alert box. Be sure that the window is the active window before using the alert box.

The 'TellFiler' procedure is also used to send pre-defined replies and unsolicited messages to the Filer. The only unsolicited messages currently defined are fcDocClosd and fcInitFailed, an abnormal termination during program initialization.

An application is also responsible for maintaining several menu items and informing the Filer via DoFilingCmd when the user invokes them. The menu layout is defined in Tesler's 'Menu Terminology' memo of May 30, 1982. The items of interest are all in the first menu:

Menu item	DoFilingCmd parameter
Close Everything on Desk	cmdCloseAll
Close "window title'	cmdClose
Save & Put Back	(see below)

When the user invokes 'Save and Put Back' the application should attempt to close the document as if it had received an fcClose Filer event. If the close is sucessful, 'TellFiler' docClosd with reason of putBack. If the close was not successful, tell the user why via alerts - DON'T TELL THE FILER.

The CopyDoc procedure is provided for application use. It copies all document diskfiles from the source prefix to the destination prefix. The number of an unbound LDSN must also be provided. The CopyDoc routine temporarily opens a large, memory-resident data segment for data transfer. The data segment is unbound and destroyed before CopyDoc returns.

As an alternative, the application can supply an bound data segment by negating the LDSN parameter. (NOTE: until additional O/S interfaces are available at 5.2, the application must also set useDsAdrs to the beginning address of the bound data segment and useDsMemSize to the data seg's length.)

```
}

CONST

    { Errors - range is 4025 thru 4049 }

    fceNoErrors =          0;  { All OK                                        }
    fceAborted =        4033;  { User type 'apple .'                           }
    fceBadEventType =   4025;  { Event type must be docOpen/Close/Copy/Terminate}
    fceBadReason =      4026;  { FReason does not match FReply                 }
    fceCantRead =       4027;  { Cannot read from the source document          }
    fceCantWrite =      4028;  { Cannot write to the destination document      }
    fceInUse =          4029;  { File opened privately or being written to     }
    fceNoMemory =       4030;  { Insufficient space for IO buffer              }
    fceOutOfDiskSpace = 4031;  { Insufficient space on distination volume      }
    fceBadLDSN =        4032;  { OS error attempting to use the LDSN provided}

    { filing menu commands - for filing menu items in app menus }

    cmdClose        = 1001;
    cmdClosAll      = 1002;


TYPE
```

```
FilingCmd = LONGINT;

FilerOp  = (fcClose,       { Update and close doc using same diskfile names  }
            fcCopy,        { Update doc into new diskfiles, source unchanged }
            fcDfClose,     { Close the diskFile for the refnum provided      }
            fcNone,        { No doc to open, i.e. user executed program      }
            fcPut,         { Update and close doc to new location (fDocName) }
            fcResume,      { Open doc and display content in window          }
            fcShred,       { Close the doc and delete the diskfiles          }
            fcSuspend,     { Close doc, keep edits seperate, save state      }
            fcTerminate);  { Terminate process, suspend any open docs        }


FReply =   (dfClosed,      { Reply to fcDfClose                              }
            dfNotClosed,   { Reply to fcDfClose                              }
            docClosd,      { Reply to fcClose, fcSuspend, fcShred            }
            docNotClosed,  { Reply to fcClose, fcSuspend, fcShred            }
            docXfered,     { Reply to fcCopy, fcPut                          }
            docNotXfered,  { Reply to fcCopy, fcPut                          }
            InitFailed);   { Unsolicited, app could not initialize           }
                           { fcTerminate reply is 'ImDying' call             }
                           { fcNone does not require a reply                 }


FReason =  (allOK,         { FilerOp completed without error or problem      }
            badData,       { Unable to display the document                  }
            cantRead,      { Unable to read in the document                  }
            cantWrite,     { Unable to write out document, disk problems     }
            dirtyDoc,      { The document was edited an may be inconsistent  }
            internalError, { Unexpected program error at any time            }
            newerDoc,      { Doc created by newer version of app             }
            noDiskSpace,   { Insufficient disk space to complete FilerOp     }
            noMemory,      { Insufficient memory for data segments, etc.     }
            noMoreDocs,    { App can't handle any more documents             }
            okButNoMore,   { FilerOp completed, but no more docs please      }
            docPutBack,    { App processed menu 'Put Back'                   }
            aUserAbort);   { User aborted filerOp                            }

FilerExt =  RECORD                        { Returned by 'GetAddParms'        }
            theFlrOp:  FilerOp;           { The requested operation          }
            thePrefix: Pathname;          { Diskfile name prefix             }
            theDF:     INTEGER;           { Diskfile refnum(fcDfClose)       }
            END;

FCopyOp  = (fcDocCopy,     { Set diskfile DTC to now, DTM to 0               }
            fcDocMove,     { Duplicate DTC, DTM values                       }
            fcDocBackup);  { Duplicate DTC, DTM. Set DTB on source diskfile  }

{*****************************************************************************}
{*                                                                           *}
{*  The following TYPEs are exported only for use by other Filer UNITs.      *}
{*  They can and should be ignored by all other users.                       *}
{*                                                                           *}
    hFilerExt = ^pFilerExt;            { EventRecord.userData as a handle}
    pFilerExt = ^FilerExt;
{*                                                                           *}
```

```
    ReplyPtr  = ^Reply;
    Reply =     RECORD                    { Redefines EventRecord.userData  }
                theReply:  FReply;
                theReason: FReason;
                END;
(*                                                                          *)
(*****************************************************************************)


PROCEDURE CopyDoc (VAR error: INTEGER; fromPrefix, toPrefix: Pathname; useLdsn: INTEGER;
                   theOp: FCopyOp; VAR docSize: LONGINT);

{ This procedure copies all diskfiles in the 'fromPrefix' document to the
  'toPrefix' document.  The document is tranferred via the dataseg bound to useLDSN.
  If useLDSN is positive, this procdure will temporarily bind its own data seg
  for the duration of the operation.  A negative useLDSN indicates that the caller
  has already bound a dataseg to useLdsn (it should be of copyDsSize if possible).
  TheOp determines how a diskfile's DTM, DTC, and DTB fields are to be set.
  Applications should always pass 'fcDocCopy'. DocSize returns the number of blocks,
  including file system overhead, occupied by the document's diskfiles.

  Errors:  fceNoErrors, fceAborted, fceCantRead, fceCantWrite, fceOutOfDiskSpace,
           fceNoMemory }


PROCEDURE DoFilingCmd (whichCmd: FilingCmd);

{  This procedure is used by an application when a filing menu item is selected. }


PROCEDURE GetAddParms (VAR error: INTEGER; theEvent: EventRecord;
                       VAR theFilerExt: FilerExt);

{  This procedure is used to access the additional paramters sent with Filer-
   related events: docOpen, docClose, docCopy, and docTerminate.  'userData'
   is the EventRecord.userData field from the received event. If the event
   type is not one of those four, the badEventType error is returned.

   errors: fceNoErrors, fceBadEventType }


PROCEDURE TellFiler (VAR error: INTEGER; what: FReply; why: FReason;
                     myFolder: WindowPtr);

{  This procedure is used by an application to send a message to the Filer.
   Usually is it used to reply to an event sent by the Filer (a reply is nearly
   always required), but it is also used to send an unsolicited message, such
   as abnormal termination, to the Filer.

   'MyFolder' is the primary window used to display the document, i.e. the
   one passed in on the docOpen event.  It can be NIL if there isn't an
   open document.

   'what' and 'why' constitute the message that your are sending the Filer:
```

IN RESPONSE TO	VALID 'WHAT'S	VALID 'WHY'S	
fcResume, fcNone		no response required if doc opened ok.	
fcClose	docClosd	allOK	
	docNotClosed	cantWrite:	disk I/O problems
		cantRead:	unable to read doc diskfiles
		dirtyDoc:	edited doc may be inconsistent
		noDiskSpace:	can't write new diskfiles
		noMemory:	machine is too small
		internalError:	application error, last resort
		aUserAbort:	user abort
fcShred	docClosd	allOK	
fcSuspend	docClosd	allOK	
	docNotClosed	cantWrite:	disk I/O problems
		cantRead:	unable to read doc diskfiles
		noDiskSpace:	can't write new diskfiles
		noMemory:	machine is too small
		internalError:	application error, last resort
		aUserAbort:	user abort
fcCopy, fcPut	docXfered	allOK	
	docNotXfered	cantWrite:	disk I/O problems
		cantRead:	unable to read doc diskfiles
		dirtyDoc:	edited doc may be inconsistent
		noDiskSpace:	can't write out new doc
		internalError:	application error, last resort
		aUserAbort:	user abort
fcTerminate	'response' is a call to 'ImDying'		
fcDfClose	dfClosed	allOK	
	dfNotClosed	internalError:	for any reason

==========

UNSOLICITED MSGS	VALID 'WHAT'S	VALID 'WHY'S	
can't display doc	docClosd	badData:	doc damaged during operation
		newerDoc:	doc version newer than app ver.
		noDiskSpace:	can't open data segs, etc.
		noMemory:	can't open data segs, etc.
		noMoreDocs:	can't open another doc. (fcNone)
		internalError:	application error, last resort.
user abort fcResume	docClosed	aUserAbort:	user pushed 'command .'
doc 'PutBack'	docClosd	docPutBack:	doc closed as user requested.
prog initlzat'n	initFailed	noDiskSpace:	can't open data segs, etc.
		noMemory:	can't open data segs, etc.

```
                                    internalError: application error, last resort.
                                    aUserAbort:    user abort
                                    DO NOT NEED TO CALL 'ImDying' after this.

  ==================================================================================

  errors:  fceNoErrors, fceBadEventType, fceBadReason }

{*****************************************************************************}
{*                                                                           *}
{* The following procedures are for use by the Filer only.                   *}
{*                                                                           *}
{*****************************************************************************}

PROCEDURE CopyDiskfile (VAR err: INTEGER; source, destination: Pathname;
                        bufrAdrs, bufrSize: LONGINT; theOp: FCopyOp;
                        VAR osErr: INTEGER);

IMPLEMENTATION

{$IFC FcSymbols }
{$D+ }
{$ELSEC }
{$D- }
{$ENDC }

{$IFC FcDebug }
{$R+ }
{$ELSEC }
{$R- }
{$ENDC }

{$I 1:FCimpl.text}

END.

{

Change Log:

 5-Oct-82 A2 Release

20-Oct-82 Added FReason 'docPutBack', updated doc for streamlined protocol

28-Oct-82 Deleted FReply 'diskFreed', 'diskNotFreed', FilerOp 'freeDisk',
          and FilerExt 'bytesReqd'.
          Added error fceAborted and CopyDoc/CopyDiskFile FCopy param.

 1-Nov-82 A3 Release
 2-Nov-82 A3 ReRelease
 5-Nov-82 Added 'aUserAbort' reason to several replies.
 9-Nov-82 A3 ReRelease
 3-Jan-83 Removed 'cmdPutBack', 'docOpened', 'docNotOpened' - support for old protocols.
 3-Jan-83 A4 pre-release
 5-Jan-83 Added 'newerDoc' to fReason
```

o The End o