APL Data Systems
a division of TRA Inc.

APL/DS FILE SYSTEM

APL Data Systems

a division of TRA Inc.

APL/DS FILE SYSTEM

710 West Main, Arlington, TX 76013 Phone (817) 265-2132

TABLE OF CONTENTS

INTRODUCTION

An APL/Z80 file is a set of APL data objects. Each file is composed of from 1 to 252 components with each component holding one data object. The component can contain any value that can be assigned to a variable, such as a scalar, a vector, or a matrix. Also, different components may contain different kinds of objects. For example, the first component of a file may contain a literal vector while the second component contains a numeric scalar.

Files offer many advantages over workspaces for storing of data.

- A file can hold up to 256,000 bytes of data depending upon the type of disk on which the file is stored.
- While only one workspace is available to the user at any one time, up to 4 files may be used simultaneously.
- Files are used via a set of file functions which can be invoked by the user or in a defined function (program).

Files are also comparable to workspaces in many respects.

- A file can be created, used to store data, and erased.
- Just as a workspace must be loaded before it can be used, a file must be "tied" before it is available for use.
- Both files and workspaces reside on disks that can be removed from disk drives.

There is a major difference between files and workspaces. A data object found in a workspace is referenced by using its name. A file component is referred to by its component number. A component number is a positive integer between 1 an 252 inclusive.

Copying File Functions into the Active Workspace

File functions are stored on a disk as copyobjects. Before they can be used to manipulate files they must be copied into the active workspace. To do this the copyobject LOADFILES is executed.

```
      )COPY LOADFILES
LOADFILES                          (system reads functions
FILE FUNCTIONS COPIED               into active ws)
```

After the function LOADFILES has loaded all the file functions and global variables, it will erase itself. LOADFILES also sets up the function F to be autostarted. The function F must always be autostarted when a workspace is loaded since it initializes global variables used by other functions. This is done automatically by this file package.

Before a disk can be used to store files, it must contain a file directory. A file directory is a N by 13 literal matrix stored as a copy object. This file directory contains the names and space allocations of files stored on a single disk. Thus each disk will have its own file directory of all files on that disk. A file directory can be placed on a disk by use of the function NEWDISK.

```
      NEWDISK
WHICH DRIVE CONTAINS THE DISK TO BE INITIALIZED ? (0,1,2 OR 3)
□:
      0
A CHECK WILL NOW BE MADE TO ASCERTAIN WHETHER A FILE DIRECTORY APPEARS ON THAT
      DISK.
******************************
THE SEARCH REVEALS THAT A DIRECTORY IS
NOT FOUND

ON THE DISK.  A DIRECTORY WILL NOW BE CREATED.  PLEASE STAND BY.
NEW FILE DIRECTORY CREATED; FILES MAY NOW BE USED.
```

At this point the disk in unit zero has an empty file directory.

Note if there are files on a disk and NEWDISK is executed, then a new

file directory will not be created. Rather, one must use FERASE to erase the files. (Wiping out the directory, but not the files leaves space used on the disk which is garbage to the user.) A sample execution of NEWDISK is:

```
      NEWDISK
WHICH DRIVE CONTAINS THE DISK TO BE INITIALIZED? (0,1,2, OR 3)
□:
      1
A CHECK WILL NOW BE MADE TO ASCERTAIN WHETHER A FILE DIRECTORY APPEARS ON
      THAT DISK.
******************************
THE SEARCH REVEALS THAT A DIRECTORY IS

ON THE DISK CONTAINING THE FOLLOWING:
VENDOR     20000
INVOIC     30000
AR2        12800
IT MAY BE ERASED BY USING 'FERASE'.  A NEW FILE DIRECTORY HAS NOT BEEN CREATED.
```

USING FILES

Files are manipulated via the file functions loaded by LOADFILES. These functions can be used to create, to read, to write, to obtain file status information, and to destroy files. Each function name starts with 'F' and indicates the type of operation that it performs. For example, FREAD reads a component of the file and returns its value. Every file has a file name. A file name can consist of from one to six alphanumeric characters, of which the first must be alphabetic. Before a file can be used, it must be "tied." Establishing a tie temporarily associates a positive integer (tie number) with a particular file. Until the file is untied, the file functions use this tie number, instead of the file name, to refer to the file.

After a file number has been associated with a particular file, the file is said to be "tied." Up to four files may be tied at a time. Each tied file in a workspace must have a unique tie number. One need not use the same tie number each time a file is tied.

The following examples may be performed if the disk contains a file directory (copyobject called "FILEDI"). If not then one must be created using the function NEWDISK.

To create a file the function FCREATE is used. The syntax of FCREATE is:

```
'filename size' FCREATE tn, dn
```

where "filename" is a one to six character alphanumeric name for the file, "size" is the size of the file in bytes, "tn" is the integer tienumber between 1 and 99, and "dn" is either zero, one, two or three specifying the disk unit on which the file is to be created.

```
'DATA 9000' FCREATE 23,0
```

This creates a 9000 byte file called 'DATA' on unit zero with tienumber 23.

The file is also tied when it is created and is ready to be used. Note that the "size" argument is optional when using FCREATE. If the size is omitted and only a filename given, then a default size of 12,800 bytes is allocated for the file.

'XYZ' FCREATE 1,0

This creates a 12,800 byte file on unit 0 with tienumber 1. Another important point to know about the sizes of files are the minimum and maximum file sizes. The minimum file size the system will allow one to create is 5120 bytes. The maximum size of a file is dependent on the type of disk media being used to store the file. Usually the user should anticipate the amount of storage needed and create files accordingly.

The function FERASE causes files to be erased. A file must be tied before it can be erased or a FILE TIE ERROR will result. The syntax for this function is:

'name' *FERASE* tn,dn

where "name" is the name of the file, "tn" the tienumber, and "dn" the disk unit containing the disk upon which the file resides. For example to erase the file DATA we created above:

'DATA' FERASE 23,0

This causes the file DATA with the tie number 23 to be erased.

The function FRENAME can be used to change the name of a file, its size, or the disk on which it resides. The syntax of FRENAME is:

'name size' *FRENAME* tn,dn

where name is the new name associated with the file with tienumber tn. The argument "size" is the size in bytes that the new file is to assume. This argument is optional, but can be used to change the size of a file that already exists. If the size is not specified then the size of the new file is the same

as that of the old one. The last argument, dn, indicates which disk unit the new file is to reside. For example to change the name of file XYZ with tie-number 1 on the disk in drive 0 to ABC:

'ABC' FRENAME 1,0

To change the name, size, and disk of the file ABC (make sure there is a disk in unit #1):

'MNO 19000' *FRENAME* 1,1

This would change the name of the file called ABC to MNO, change its space allocation to 19,000 bytes, and copy the file from the disk in unit #0 to the disk in unit #1. FRENAME can also be used sometimes to collect garbage in files that appear to be full and thereby free up additional space (see Appendix B).

Files that were created in a previous session must be tied before they can be used in subsequent sessions. The function used to tie files is called FTIE and its syntax is the same as FERASE above. Assuming the file 'DATA74' already exists, then

'DATA74' *FTIE* 99,0

ties the file name DATA74 with a tie number of 99 on disk unit 0. Also note that if one attempts to tie a file which has already been tied, then the function FTIE will simply drop through without flagging an error. Thus if we attempted to tie the file DATA74 again:

'DATA74' FTIE 74,0

no error would result.

To untie the same file we would use the function

FUNTIE 99

Note that only the tie number was used to untie the file. FUNTIE can also be used to untie several files at the same time. If four files were tied with

the tie numbers one through four then,

FUNTIE 1,2,3,4

would untie all four files. Files are untied if the workspace is either cleared or if the)OFF command is executed. The only file functions that require the file name as a left argument are FCREATE, FERASE, and FTIE. FLIB, FCREATE, FERASE, and FTIE are also the only functions requiring that the disk unit be specified. All other file functions use only the tie number to refer to a file.

There are three functions which are used to place values in a file and to retrieve values. To append values to the end of a file, the function FAPPEND is used. FAPPEND takes as a right argument the tie number of a file, and as a left argument a value to be placed at the end of the file. This value may be a variable or an expression. To place a person's name in the first component, age in the second component and telephone number in the third component would require:

```
'JOE' FAPPEND 1
23 FAPPEND 1
273 2381 FAPPEND 1
```

Thus, we have stored a literal vector, a scalar, and a numeric vector in components 1,2, & 3 respectively. A schematic representation of the file MNO is:

Components	1	2	3
Contents	*JOE*	23	273 2381

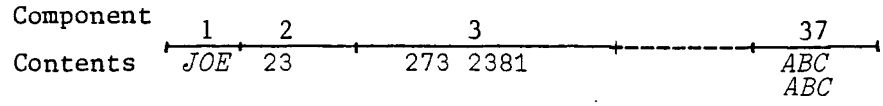
The function FAPPEND discussed earlier always places values at the end of the file. To randomly write values into a file the function FWRITE is used. The syntax is:

value *FWRITE* tienumber, componentnumber

For example, suppose after writing into the first three components of MNO above, one wished to place a 2 by 3 literal array in component number 37. The function FWRITE is used.

```
(2 3ρ 'ABC') FWRITE 1,37
```

Note that only components 1 through 3 and 37 now have values. The file represented schematically is:



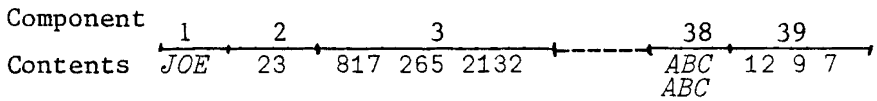
FWRITE can also be used to write new values into already existing components. If for example the phone number in component number three should be 817 265 2132 instead of 273 2381, we could correct the error by:

```
817 265 2132 FWRITE 1,3
```

Now there would be a new value in the third component. Remember also that FAPPEND places values at the end of a file, so that if one wishes to add information in components after no. 38, FAPPEND can be used. Thus,

```
12 9 7 FAPPEND 1
```

would place the vector 12 9 7 in component 39.



Sometimes when a FILE FULL error is received after using FWRITE or FAPPEND, FRENAME can be used to collect garbage in the file (thus obtaining more space) as indicated below.

```
      'ABCDEFG' FAPPEND 1
FILE FULL ERROR
      'MNO' FRENAME 1,1
      'ABCDEFG' FAPPEND 1
```

Or the file space could be increased using

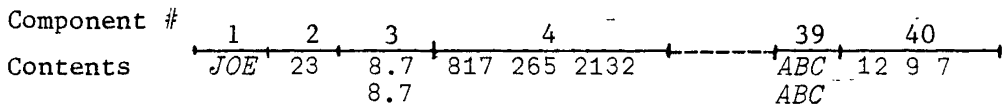
```
'MNO 25000' FRENAME 1,1
```

Note that earlier the file MNO was increased to a size of 19000 bytes.

The function FWRITE can also be used to insert a new component between two consecutive components. For example to place a new data object in a component between components two and three we would enter:

```
      (2 1ρ8.7) FWRITE 1,2.5
```

This would cause the matrix to be inserted in the file. Note that the new component number would not be 2.5, but rather it would be 3. Also higher numbered components would be incremented by one. So the file would now be depicted as:



If we wanted to read the age from the second component of the file then the function FREAD is used. Its syntax is FREAD tienumber, component number.

```
      FREAD 1,2
23
```

Note that FREAD produces an explicit result. Thus, the values in a component can be assigned to a variable

```
      A←FREAD 1,4
      A
817 265 2132
```

or used directly in an arithmetic expression.

```
      4+FREAD 1,2
27
```

If we try to read a component that has not yet been specified, then an error will result

```
      FREAD 1,14
FILE INDEX ERROR
```

File functions can also be used within user defined functions. The following program places literal vectors in user specified components of a file.

```
      ∇FUNCTION
[10]   'FILE' FCREATE 3,0
[20]   LOOP: 'ENTER STREET ADDRESS.'
[30]   DATA←⍞
[40]   →EXIT×⍳0=ρDATA
[50]   DATA FWRITE 3,□
[60]   →LOOP
[70]   EXIT:FUNTIE 3
      ∇
```

The function FTIE may be used as the first statement of every function which accesses files. If the file is already tied, FTIE will pass control to the next line of the function.

The function FDELETE is used to delete indicated components from a file. Its syntax is:

FDELETE tienumber, component number(s)

FDELETE 3,4

will eliminate components three and four from our file.

Component	1	2	...	39	40
Contents	*JOE*	23		*ABC* *ABC*	12 9 7

FDELETE does not renumber other file components.

Several other functions also present allow a user to obtain file status information. To obtain a list of existing files and their creation sizes (in bytes) the FLIB function may be used. The FLIB function requires a right argument specifying a disk unit and returns an N by 15 literal matrix:

```
      FLIB  0
DATA74  12800
FILE    12800
      FLIB  1
MNO     25000
```

One may also determine which files have been tied and the corresponding tie and unit numbers. The function FNAMES returns an N by 6 literal matrix containing the names of all files which are currently tied. To obtain the tie numbers of all currently tied files the FNUMS function is used. FNUMS returns a N by 2 numeric array whose values correspond to the file names given by FNAMES. FNAMES and FNUMS require no arguments, and both return an explicit result.

```
      FNAMES
MNO
      FNUMS
1  1
```

So the file named NMO has a tienumber of 1 and the disk upon which it resides is in disk unit #1.

To determine the upper and lower component limits of a file the FLIM function is used. The syntax for FLIM is

FLIM tienumber

The FLIM function returns a numeric vector of length two. The first component of this vector is the number of the first file component in use and the second is one more than the number of the last. If a file has just been created and does not contain any components then FLIM returns a 1 1.

For example, FLIM applied to the file MNO:

```
      FLIM 1
1 41
```

Note that not all the components between 1 and 41 contain data.

The function FCUSED (file components used) may be invoked to ascertain file components which are in use. The FCUSED function requires the file tie number as a right argument. For example, to determine the components used in our file called MNO which is tied by the number 1:

```
      FCUSED 1
1 2 39 40
```

The user should be aware that the file system uses no additional space when empty components are placed within the limits of a file. That is, if the information in our MNO file was in components 1, 2, 3 and 4 rather than in components 1, 3, 39 & 40, there would be no saving in space. The maximum number of components in a file cannot exceed 252; moreover, the largest value for a component number is 252.

APPENDIX A

ERROR Messages

FILE TIE ERROR	-Tie number has already been used to tie another file or file has not yet been tied.
FILE INDEX ERROR	-Component(s) specified does not exist or is not between 1 and 252.
FILE NAME ERROR	-File name specified is already in use or file does not exist.
FILE TIE QUOTA USED UP	-An attempt has been made to have more than 4 files tied at once.
FILE RESERVATION ERROR	-There is no more room on the disk for new files.
LENGTH ERROR	-Disk unit not specified in right argument to FCREATE, FTIE, or FERASE.
DISK SPEC ERROR	-File is not on the disk specified
I/O ERROR or *FILESUBSYSTEM ERROR*: *JUNCTION* [3]	-An I/O error has occurred. Try the operation again. If the error persists then there is a hardware I/O problem.
BDOS ERR ON × :	-A nonexistent disk drive has been selected. The contents of the workspace will be lost. Press the reset button in order to reload the APL system.

Whenever a file function returns an error it sets the global variable called *FERR* to a non-zero value and returns to the calling program or calculator mode. When the file system functions are used as subroutines the value of *FERR* can be tested to determine if the operation was completed successfully.

Note that if FERR is set to a nonzero value the user must reset the value of FERR before using another file function.

System Errors

The APL/DS file system is a set of programs which interface with the processor 108 file processor. The functions check to see that operations are performed correctly and that information exchanges are completed successfully. When a function detects an error it will print a message in the form:

FILE SUBSYSTEM ERROR: function name [x]

where function name is the name of the function where the error occurred. X will be a number indicating one of the errors below:

X	ERROR
1	Invalid SIGN-ON INFO in data or control variable
2	No more disk space or allocated disk space used up
3	I/O error
4	Workspace full
5	Wrong length record
6	Attempt to specify control variable to be other than integer between 0 and 252
7	File not found
8	Attempt to write past record 252
9	End of file or attempting to read unwritten record in random access

If a file system error should occur please report it along with relevant information to APL/DS.

APPENDIX B

Workspace Environment for Using Files

There are a few rules that must be followed by the user of the file system. First, before the user copies any file functions into a clear workspace, it is suggested that s/he increase the size of the symbol table using the)SYMBOLS system command. This will allow adequate symbols for the file functions and any application programs. Also several file functions may be erased if they are not used. The function F must also be autostarted every time the workspace containing file functions is loaded since it sets up the global variables TIENUMBERS and TIENAMES.

When a file is created or tied, two shared variables are created for use by the file system. The variables are called CONTROLn and FILEn where n is the tie number of the file. These variables must not be altered or erased or a file subsystem error could result.

FILE FULL Errors and Empty Files

It is possible for an APL/DS file to be empty and yet not be able to hold any data. The system attempts to work around this but it is suggested that the following rules be followed.

1) Do not erase files by deleting all the file components. Rather, erase the file.

2) If a FILE FULL error does occur try using FRENAME to increase the size of the file. In some instances, executing FRENAME specifying the same name, same size, and same unit will solve the problem. If the disk is full you may need to specify the same name, same size, and a different unit.

File Function Dependencies

In most applications only a few of the file functions will be used routinely. As a consequence, many users will want to maintain in a workspace only those file functions which are regularly used. The APL/DS file package uses 11,800 bytes immediately after LOADFILES is executed. However, when only FREAD and FWRITE and the requisite functions and variables are in the workspace, only about 2,700 bytes are utilized. FRENAME alone requires almost 2,000 bytes, and should be copied into the workspace only when required.

The file functions which require other functions are listed below along with the functions that must also reside in the workspace for them to execute.

FILE FUNCTIONS	*FUNCTIONS SUPPORTING THOSE ON THE LEFT*
FCREATE	*FLIB*
FERASE	*FLIB*
FTIE	*FLIB*
FWRITE	*FCOMPONENTMAP*
FAPPEND	*FCOMPONENTMAP, FWRITE*
FREAD	*FCOMPONENTMAP*
FDELETE	*FCOMPONENTMAP*
FLIM	*FCOMPONENTMAP*
FCUSED	*FCOMPONENTMAP*
FRENAME	*FCREATE, FTIE, FLIB, FCUSED, FWRITE, FUNTIE, FCOMPONENTMAP, FNAMES, FERASE, FREAD*

Remember also that the function F must be autostarted (latent expression) every time a workspace is loaded in which files are to be used. The function LOADFILES sets up this expression. A copy of LOADFILES appears on page 17. It may be modified to copy the functions required into workspaces. If applications packages are developed which maintain selected file functions in the package workspace, the function F must be maintained in that workspace in addition to the selected file functions and their support functions. That is, if FREAD and FWRITE are needed in a workspace, then FREAD, FWRITE, FCOMPONENTMAP, and F should reside in the workspace.

APPENDIX C

Rebuilding the File Directory

If a disk which is being used to store files loses its filedirectory, then those files cannot be accessed. This can be corrected by performing several steps.

1) Place disk in the system drive
2) Return to the host operating system.)OFF
3) Execute the DIR command at the host OS level and make a list of file names
4) Reload the APL system (you may have to switch disks in the system drive for this)
5) Replace disk needing filedirectory in system drive (if removed in 4)
6) Insert the APL/DS file disk in drive 1 and address drive 1
7) Copy and execute the copyobject 'FFIXFD.'

These steps should allow one to rebuild the filedirectory so that the FLIB functions return the same file names at the APL level that the DIR command does at the host OS level.

APL/DS FILE SYSTEM

FILE FUNCTIONS

SYNTAX		USE
	LOADFILES	COPY FILE FUNCTIONS
	NEWDISK	INITIALIZE A DISK FOR FILES
'filename size'	*FCREATE* fn dn	CREATE & TIE A NEW FILE
'filename'	*FERASE* fn dn	ERASE A FILE
'filename'	*FTIE* fn dn	TIE A FILE
	FUNTIE fn fn fn	UNTIE A FILE(S)
expression	*FWRITE* fn cn	PLACE THE EXPRESSION IN A SPECIFIC FILE COMPONENT
expression	*FAPPEND* fn	PLACE THE EXPRESSION IN THE NEXT HIGHEST FILE COMPONENT
	FREAD fn cn	READ COMPONENT cn FROM FILE fn
	FDELETE fn cn cn cn	DELETE COMPONENTS cn FROM FILE fn
result ←	*FLIB* dn	GIVES FILE NAME & RESERVATION SIZE
result ←	*FNAMES*	GIVES THE NAMES OF TIED FILES
result ←	*FNUMS*	GIVES THE FILE NUMBER & DRIVE NUMBER OF TIED FILES
result ←	*FLIM* fn	GIVE FIRST & NEXT HIGHEST FILE COMPONENT
result ←	*FCUSED* fn	GIVES NUMBERS OF FILE COMPONENTS USED
'filename size'	*FRENAME* fn dn	RENAME OR RESIZE A FILE

fn: FILE NUMBER
cn: COMPONENT NUMBER
dn: DRIVE NUMBER (0,1,2 or 3)

APL DATA SYSTEMS
(A division of TRA, Inc.)
710 West Main St.
Arlington, TX 76013