Disc Access

TODO - "Update conventions and procedures of this whole section for AOZ where necessary"

Disc drive names

Each disc drive used by your Amiga is identified by a simple three-character code, followed by he colon character to distinguish the name of the drive from a file name. The internal floppy disc drive is referred to like this:

```
Df0:
```

If you have installed additional floppy drives, they will be named Df1: then Df2: and so on. I lard drives are identified by a similar code, with the first hard disc drive carrying a zero, the second a one, and so on, like this:

```
Dh0:
```

Volume names

The Amiga is happy to refer to an individual disc by name instead of looking for the disc drive code, as long as the string of characters that make up the name of the disc carries the colon character, as follows:

```
AMOS_PROFESSIONAL:
```

The titles of "discs are known as "volume" names, which is the equivalent of the title of a written volume in a library. AOZ automatically checks each available drive for the required disc, and if it cannot be found, the "Device not available" error will be given.

Whenever a new disc is prepared for use via the Workbench, it is automatically given the name "Empty", waiting for you to re-name it with a suitable volume title, after clicking on the I Rename] option. It is very bad practice to give the same name to more than one disc, as both the Amiga and its operator can get confused by sloppy naming. If different discs do have the same volume name for any reason, you will have to refer to the appropriate drive name to
tell AMOS Professional precisely which of these discs you are interested in. For example:

```
Dir "Df0:"
```

The DIR command is used to print out a directory index of a disc, and is explained below.

Files and directories

If you think of a disc as a self-contained "volume" , then that volume can contain one or more "folders" of information, and each folder can hold all sorts of "files". Before any file can be accessed and used, it has to be found in the file directory of its disc. The next section of this Chapter explains how files are managed with AOZ, but first you should be aware of the set of objects known as "logical devices".

Logical devices are used by the Amiga's operating routines to work out the exact position of important system files, such as the fonts used for text characters and the device handlers used for peripherals. Each device is normally assigned to a specific directory on the current start-up disc.

For example, the directory containing the current fonts used by AMOS Professional is called FONTS: whereas the SAY command uses a library file that can be examined by typing the following line from Direct Mode:

```
Dir "Libs:"
```

DIR

instruction: print directory of the current disc

```
Dir
Dir path$
```

The DIR command is used to examine the directory of the current disc and list all of its files on screen, like this:

```
Dir
```

Any folders in the listing will be distinguished by a leading asterisk character *. The listing can be stopped at any time by pressing the [Space-bar] and then started again in the same way. Note that if you change discs without informing AMOS Professional and then try to get a directory listing, you may be presented with a system requester.

The simple solution is to re-insert the requested disc and try again.

DIR/W

instruction: print out directory in two columns

```
Dir/W
Dir paths$ /W
```

This command performs exactly the same task as DIR, but displays the list of files in two separate columns across the screen. So by using this double width, twice as many filenames can appear on screen at any one time.

There is no need for DIR or DIR/W to list every file on the disc. Certain files or groups of files can be extracted by specifying optional "pathways", so that only files which satisfy a certain set of conditions are listed.

The broadest of these paths gives the name of the disc or the drive to be examined. A colon must be added to the disc name, like this:

```
Dir "FONTS:"
Dir "Dh0:"
```

The next selective category that can be defined is a single folder of filenames to be listed. For example:

```
Dir "Objects/"
Dir "AMOSPro_Examples:Objects/"
```

The pathway for a listing can be further narrowed by requesting that only the filenames that satisfy certain conditions will be printed, and that each character in the filename must match the characters in your request exactly.

If you wish to make a more general search, you can use the asterisk character "*" to be regarded as a substitute for any list of characters in a filename, up to the next control character. For example, a file named "Music" will be searched for if you command this:

```
Dir "Music"
```

But the use of an asterisk would broaden the search:

```
Rem List all files starting with M
Dir "M*"
```

That could give the following directory listing:

- Music
- Megalomania
- Milk

As a default, this option ignores any files that include extensions of the type used by MS-DOS, such as "Mad.Asc". The full stop character "." is used to match a filename extension, and is often used with the asterisk character to list all the files in a directory with a particular extension, like this:

```
Dir "Music.*"
Dir "*.Megalomania"
Dir "*.*"
```

The final narrowing of a search path is to use the question-mark character "?" to match up with any single character in a filename. For example:

```
Dir "EUROP????"
```

That would list the following filenames, if they were in the current directory:
EUROPRESS
EUROPEANS

But it would ignore the following filenames, either because the first five characters do not match, or the length of the name is different from the specified total of nine characters:
EUROPRESSES
EUROPE
EURIPIDES

Because certain filenames are too long to fit neatly in a display listing, particularly if the DIR/W option is in use, there is a simple way of setting the style of directory commands.

LDIR

LDIRAN

instructions: output directory of current disc to printer

```
Ldir
Ldir path$
Ldir/W
Ldir path$ /W
```

These two commands are used in exactly the same way as DIR and DIR/W, as explained above, and they list the directory of the current disc to a printer.

SET DIR

instruction: set directory style

```
Set Dir number
Set Dir number,filter$
```

This command must be followed by a number ranging from 1 to 100, which sets the number of characters to be displayed from each filename. There is no effect on the names themselves, only on the way they are displayed. For example:

```
Set Dir 6
Dir
```

An optional string may be added to a SET DIR command, which has the effect of filtering out pathnames from the directory search. All filenames that match up with this filer will be completely ignored. Supposing a directory began like this:

- AMO.IFF
- AMAL
- AMAT.IFF
- AMINIBUS
- AMINOACID
- AMENSROOM.IFF
- AMULET

SET DIR may now be used to restrict the display to three characters, as well as filtering out any IFF files, as follows:

```
Set Dir 5,".IFF"
```

The first two characters displayed are folder markers, this is why the value five is used instead of three.

That line would result in this amended display:

- AMA
- AMI
- AMI
- AMU

It is possible to ignore several file-paths at once, as long as each name is terminated with a single oblique character "/". For example:

```
Set Dir 8,"*.AMOS/*.IFF/*.Abk"
```

DIR$

function: change the current directory

```
s$=Dir$
Dir$=s$
```

The DIR$ function is used to contain the directory that will be used as the starting point for all future disc operations, such as loading and saving.

```
Print Dir$ : Rem Print out current directory
Rem Set directory to folder
Dir$="AMOSPro_Examples:IFF/"
```

DIR$ is similar to the CD command from the CLI, with the advantage of allowing you to read the directory as well as change it. All directories are assumed to be relative to the directory in current use, and AOZ will only search the current directory for a folder. To avoid this problem, include the name of your disc as in the above example, or use the drive name as follows:

```
Dir$="Df0:IFF/"
```

PARENT

instruction: negotiate a path through current directory

```
Parent
```

Because directories can be "nested" inside one another, files can be organised according to a range of categories.

Although this is very convenient, it is not difficult to get lost in a maze of nested files. In the following example, the folder named FOLDERA is stored in the main directory (the "root" directory) and can be regarded as the "parent" of FOLDERB and FOLDERD. Similarly, FOLDERB is the parent of FOLDERC:

- FOLDERA/
- FOLDER B/
- FOLDERC/
- FOLDERD/

The effect of the PARENT command is to load the current directory with the parent of the present folder you are using. By using this command repeatedly, you are able to get back to the original root directory simply and quickly. For example:

```
Dir$="AMOSPro_Examples:Objects/"
Dir
Parent
Dir
```

ASSIGN

instruction: assign a name to a file or device

```
Assign "Name:" To "New_Pathname"
Assign "Name:" To "Device"
```

In the original AMOS system, you were obliged to go back to AmigaDOS every time that particular directories needed changing, for example, when changing the font directories. The ASSIGN command has been provided to solve this problem, and is fully explained in Chapter 11.1.

Checking for the existence of a file

It is possible to keep a tidy mind and a tidy desk, and maintain up to date records on discs. On the other hand, you may be normal. AOZ provides three, ways to check for elusive files.

EXIST

function: check if specified file exists

```
value=Exist("filename")
```

EXIST looks through the current directory of filenames and checks it against the filename in your given string. If the names match, then the file does exist and a value of -1 (true) will be reported, otherwise 0 (false) will be returned.

As well as checking for individual filenames, even if an idiotic name is given, EXIST will search for discs and devices as well. For example:

```
Print Exist("An idiotic name")
Print Exist("DEMO:") : Rem Is a disc named DEMO available
Print Exist("Df1:") : Rem Is the second floppy drive connected
```

It is advisable to test for empty strings ("") separately, like this:

```
F$=Fsel$("*.IFF"," ","Load an IFF file")
If F$="" Then Edit : Rem return to editor if no file chosen
If Exist(F$) Then Load Iff F$,0
```

DIR FIRST$

function: get first file that satisfies current path name

```
file$=Dir First$(path$)
```

This function returns a string containing the name and the length of the first file on the current disc that matches up with your chosen search path. For example, the next routine reports the first file or folder in the current directory, followed by the first IFF file in the directory. Obviously, this could be the same file.

```
Print Dir First$("*.*")
Print Dir First$("*.IFF")
```

When DIR FIRST$ is used, the whole directory listing is loaded into memory, so you can continue to discover the name of the next file in the current directory with the following function.

DIR NEXT$

function: get next file that satisfies current path

```
file$=Dir Next$
```

Use this to return the filename that comes after the file or folder found by the previous DIR FIRST$ search. If there are no more files to come, an empty string will be returned, "". Once the last filename has been found, AOZ will automatically grab back the memory used by the directory array, and release it for the rest of your program to use. The next example prints every file in the current directory.

```
F$=Dir First$("*.*")
While F$<>""
    Print F$ : Wait 50
    F$=Dir Next$
Wend
```

Selecting a file

FSEL$

function: select a file

```
f$=Fsel$(path$)
f$=Fsel$(path$,default$,title1$,title2$)
```

This file selection function allows you to choose the files you need directly from a disc, using the standard AOZ file selector. In its simplest form, it operates like this:

```
Print Fsel$("*.IFF")
```

The string held within the brackets is a path that sets the searching pattern, in that case an IFF file. The following optional parameters may also be included:

- The optional default string is used to choose a filename that will be automatically selected if you press [Return] and abort the process.
- Title$ and title2$ are optional text strings that set up a title to be displayed at the top of your file selector. For example:

```
F$=Fsel$("AMOSPro_Examples:Objects/*.Abk")
If F$="" Then Edit : Rem Return to editor if no file selected
Load F$: Rem Load file and display first Bob
Flash Off : Bob 1,100,100,1 : Get Bob Palette : Wait Vbl
```

Naming files

To create a new folder that can be used to hold files of data, a suitable disc should be ready in the appropriate drive.

MKDIR

instruction: create a folder

```
Mkdir filename$
```

This makes a new folder on the current disc, and gives it the filename of your choice. For example:

```
Mkdir "Df0:MARATHONMAN"
Dir
```

RENAME

instruction: rename a file

```
Rename oldname$ To newname$
```

This command is used to change the name of an existing file. If your choice of new filename is already in use by another file, the appropriate error message will be given.

```
Rename "Ancient" To "Modern"
```

Running programs from disc

RUN

instruction: execute an AMOS Professional program

```
Run
Run file$
```

As well as the [Run] or [F1] facility for executing programs from the Edit Screen, the RUN command may be used on its own from Direct Mode.

When followed by a filename and used inside a program, the RUN command is extremely useful. Authors of vast computer games, involving many levels of play, need not be restricted by the storage space of a single disc or the memory available in your Amiga. Each level of play can be written as a separate program and then saved as a different filename. This means that at the end of one level of play, the next stage can be loaded from disc automatically.

For example:

```
Run "Next level.AMOS"
```

This method is known as "chaining" programs together. When programs run like this, data screens and banks will be kept, allowing you to pass data and display a screen of graphics while the next level is loading. But the redundant last program will be erased to make room for the new program, so you should remember the fact that any variables will be lost in the process.

In fact, AOZ does allow you to pass variable data from one program to another, by making use of "Command Lines".

COMMAND LINE$

reserved variable: transfer parameters between programs

```
c$=Command Lines$
```

Data for hi-scores, messages, names and so on can be carried through to the next level of computer game by the following method.

Type in the next example program:

```
Rem Program 1
Screen Open 0,640,200,4.Hires
Rem greetings sent by previous program
Print "Greetings from Program 2:";Command Line$
Input "Please type in a greeting!";A$
Command Line$=A$
Print "Running Program 2!" : Wait 100
Run "Program2.AMOS"
```

Now save that example on a suitable disc, and name it "Program1.AMOS". Next, change that example program as follows:

```
Rem Program 2
Screen Open 0.320,200,4,Lowres
Rem Greetings sent by previous program
Print "Greetings from Program 1:";Command Line$
Input "Please type in a greeting!";A$
Command Line$=A$
Print "Running Program 1!"
Wait 100
Run "Program1.AMOS"
```

Save Program 2, and call it "Program2.AMOS". Now run Program 2, which should still be in memory. After the first blank communication, the two programs will greet one another until you break into their conversation with [Ctrl]+[C].

Disc space

DFREE

function: report free space on disc

```
f=Dfree
```

This simple function returns the amount of free space remaining on the current disc, measured in bytes.

DISC INFO$

function: report free space of a named device

```
information$=Disc Info$("Name")
```

This function is used to return the amount of free space in the specified device. The string that is returned contains the name of the disc, followed by the amount of free space. Here is an example which splits the string:

```
A$=Disc Info$("Df0:")
C=Instr(A$,":")
N$=Left$(A$,C)
A$=A$-N$
D=Val(A$)
Print "Name of the disc=";N$
Print "Free space=";D
```

KILL

instruction: erase a file from current disc

```
Kill filename$
```

Be extremely careful with this command. It obliterates the named file from the current disc, once and for all. The file that is erased with this command cannot be retrieved. Kill "Permanently"

Disc files

Files are simply packages of information stored together at a particular location on disc. Each file is assigned its own name, which may contain anything from 1 to 255 characters.

Before a file can be used, it must be initialised using the OPEN IN, OPEN OUT or APPEND commands, which are explained below. When a file is opened, it must be assigned a channel number, ranging from 1 to 10. This number will be used in all subsequent disc operations to identify the file you are currently working with.

Your computer uses two types of disc files: "sequential" files and "random access" files. Here is how AOZ exploits them fully.

Sequential files

A sequential file is one that allows you to read your information only in the sequence in which it was originally created. Normally if you need to change a single item of data in t lie middle of a sequential file, you must call up that file from disc, read the whole file up to and including the item of data you want to alter, change the data and then write the whole file back to the disc.

AOZ lets you have access to sequential files either for reading data, or for writing it, but never for both at the same time. Before the theory is explained, here is some practice. Type in this example, which opens a file called "sequential.one", allows you to input some data, then closes the file:

```
Open Out 1 ,"sequential.one"
Input "Please tell me your name ";N$
Print #1,N$
Close 1
```

Now the information stored in that file can be read back, as follows:

```
Open In 1 ,"sequential.one"
Input #1,N$
Print "I remember you! Hello ";N$
Close 1
```

Every time you want to access a sequential file, it must be opened, then the information can be accessed, then the file must be closed. Those three steps must be done in exactly that order. Here is the list of commands you can use for handling sequential files.

OPEN OUT

instruction: open a file for output

```
Open Out channel,filename$
```

Use this command to open a sequential file, ready for data to be added to its end. Give the channel number and filename, as explained above. If the file already exists, it will be erased.

APPEND

instruction: add data to an existing file

```
Append channel,filename$
```

This works like OPEN OUT, but it allows you to add to your files at any time after they have been defined. If the filename already exists, your new data will be appended to it, in other words it will be added to the end of that file.

OPEN IN

_instruction__: open a file for input

```
Open In channel,filename$
```

Use this command to prepare a file so that data may be read from it. If the filename does not already exist, AOZ will report a "File not found" error.

CLOSE

instruction: close a file

```
Close file number
```

You must remember to always CLOSE a file after you have finished with it. If you forget to do this, any changes that have been made to the file will be lost.

PRINT #

structure: print variables to a file or device

```
Print #channel,variable list
```

This command is used in the same way as a normal PRINT instruction, but instead of printing information on screen it puts that information into one of your files. Simply specify the channel number to be used, then the variables you want to print out to the file. Remember to close the file's channel number afterwards, like this:

```
Open Out 2,"sequential.two"
Print #2,"Just testing"
Close 2
```

As with PRINT, the PRINT # command can be abbreviated to ? #.

***INPUT##*

structure: input variables from a file or device

```
Input #channel,variable list
```

INPUT # reads information from either a sequential file or a device such as the serial port (see OPEN PORT in Chapter XX.X) (10.4), and loads these values into a set of variables. As with the normal INPUT command, each value in the list must be separated by a comma. Additionally, every line of data needs to be ended by its own line feed character, which is the equivalent of the [Return] pressed when a line is entered from the keyboard. For example:

```
Open In 2,"sequential.two" : Rem Open file created by previous example
Input #2,A$
Print A$
Close 2
```

LINE INPUT#

structure: input variables not separated by a comma

```
Line Input #channel,variable list
```

This function is identical to INPUT #, except that it allows you to separate your list of data using a carriage return sequence, instead of the standard comma.

When reading text documents, LINE INPUT # is always recommended, because the commas used in normal written English will be treated as separators by the INPUT # structure.

SET INPUT

instruction: set end-of-line characters

```
Set Input code1,code2
```

SET INPUT is used to set which characters you want to input to end a line of data. Many computers need both a [Return] and [line feed] character at the end of each line, but the Amiga only needs a [line feed]. This means that if you wanted to import files from an ST via the serial cable, for example, unwanted [Return] characters would litter your input.

SET INPUT solves this problem by allowing you to select two Ascii values as your end-of-line characters. If you prefer to use a single character only, make the second value a negative number. For example:

```
Set Input 10,-1 : Rem Standard Amiga format
Set Input 13,10: Rem ST compatible format
```

INPUT$

function: input a fixed number of characters from a device

```
i$=Input$(file,count)
```

Use this function to input a set number of characters from a device or file. The parameters in brackets refer to the filename or device, followed by the count of characters to be input.

EOF

function: test for end of file

```
flag=Eof(channel)
```

This tests to see if the end of a file has been reached at the current reading position, returning -1 for yes and 0 if this has not happened.

LOF

function: give length of an open file

```
length=Lof(channel)
```

LOF returns the length of an open file, and it would be pointless to use this function with devices other than the current disc.

POF

reserved variable: hold current position of file pointer

```
position=Pof(channel)
```

This changes the current reading or writing position of an open file. For example, the following line sets the read/write position to 1,000 characters past the start of the file:

```
Pof(1)=1000
```

Because disc drives are inherently random, this may be used to provide a crude form of random access with sequential files.

Random access files

AOZ takes full advantage of the second type of file used by the Amiga. Random access files are extremely useful, because they allow the programmer to access data stored on a disc in any random order. A random access file is made up of units of data called "records", and each record has its own identification number. Every record can be split up into as many smaller sections as required, with every section becoming a "field". Each field is used to hold a single item of data.

The main difference between sequential files and random access files is that you must tell AMOS Professional the maximum size of a field in advance, before you can make use of it.

A field can hold many forms of data, like a password, an invoice number or even a literary quotation. Supposing you want to create an electronic phone book. You could choose the following fields, with the following maximum number of characters in each:

Field	Max. length
SURNAME$	20
F1RSTNAME$	15
TEL$	10

Once the fields have been planned, the structure for your electronic database can be set up using the following commands.

OPEN RANDOM

instruction: open a channel to a random access file

```
Open Random channel,filename$
```

This command is used to open a channel to a random access file, like this:

```
Open Random 1,"ADDRESS"
```

FIELD

instruction: define a record structure

```
Field channel,length1 As field1$,length2 As field2$
```

FIELD$ should be used immediately after OPEN RANDOM to define a record that will be used for a random access file. This record can be up to 65535 bytes long. After selecting the channel number, give the maximum number of characters you will cater for in a field, followed by its name, then repeat the process as necessary. For example:

```
Field 1,20 As SURNAME$,15 As F1RSTNAME$,10 As TEL$
```

You can now place some records in the strings that have been set up by the FIELD command, like this:

```
SURNAME$="Professional"
FIRSTNAME$="AMOS"
TEL$="0625859333"
```

PUT

instruction: output a record to a random access file

```
Put channel,record number
```

Once a record has been placed in a string, it can be moved from the computer's memory into a record number of your random access file. If you were still using channel 1, your first record would be put into the random access file
like this:

```
Put 1,1
```

The next record will become number 2, and so on until you fill up your telephone book. Here is a simple working example. When you have created enough records, type in "exit" when prompted to enter another name.

```
Open Random 1,"ADDRESS"
Field 1,25 As NAME$,12 As TEL$
INDEX=1
Do
    Input "Enter a name:";NAME$
    If NAME$="exit" Then Exit
    Input "Enter the phone number:";TEL$
    Put 1,INDEX
    Inc INDEX
Loop
Close 1
```

Having created your phone book, you will want to use it.

GET

instruction: read a record from a random access file

```
Get channel,record number
```

This instruction reads a record stored in a random access file, after being told which channel to use and the number of the record to read. To read the first record you would use this:

```
Get 1,1
```

GET then loads this record into your field strings, and these strings may be manipulated as you like. Obviously you can only GET record numbers that have been PUT onto the disc.

Now try this example:

```
Open Random 1,"ADDRESS"
Field 1,25 As NAME$,12 As TEL$
Do
    Input "Enter Record Number: ";INDEX
    If INDEX=0 Then Exit
    Get 1,INDEX
    Print NAME$ : Print TEL$
Loop
Close 1
```

Included files

The AMOS Editor cannot rationalise your source code around the entire memory of the Amiga. This
means that if you are editing an extremely long program, the insertion of a line can be tedious. It can take a few seconds to move the memory around before allowing the next line to be inserted.

To assist the editing of lengthy programs in assembly language or C, an Include facility is provided. AOZ programmers can enjoy exactly the same benefit!

INCLUDE

instruction: specify a file for inclusion when testing a program

```
Include "File_To_Include.AMOS"
```

The INCLUDE command must occupy a line on its own, otherwise the specified file will not be detected , and so it will not be included. The effect of INCLUDE on a file is as follows:
Immediately before a program is tested, AOZ scans the beginning of each program line for an INCLUDE instruction.

If an INCLUDE is encountered, AMOS Professional opens the specified file, reads its length and checks its validity.

This process takes place for each INCLUDE that is found, in order.

A memory buffer is reserved for the total length of the re-created program.

AOZ now copies sections of the source program, without the Includes, and loads the files from disc.

All files are now closed, and with the memory buffer holding the re-created program, the testing process begins as normal.

You will need enough memory to hold the original buffer space and the included files at the same time for this process to operate, but if your program is large enough to slow down the Editor it is obvious that you have access to a reasonable amount of memory.

Please note that included files are only supported in the original source, and an INCLUDE in an included file will generate an error when the program is run. The re-created buffer is erased as soon as the program is left, so the specified files must be loaded every time the program is tested. If programs are included which have memory banks, these banks will be left out.

IBM and ST users

The commercially available Cross Dos package allows AMOS Professional to access discs in IBM-clone format or Atari-ST format. Discs that are in either of these formats are identified by a three-character code of the two letters "Di" followed by the number of your drive. So an ST format disc in the Amiga's internal drive would be named as follows:

```
Di0:
```

Because AMOS Basic evolved from STOS (Atari) Basic, every effort has been made to help STOS users convert their programs to AOZ. STOS programs should be saved to disc in Ascii format using the [FSAVE] "*.ASC" option. This disc should be inserted into an Amiga floppy disc drive that has been mounted by Cross Dosas an IBM drive.

Certain STOS programs will need modification before they will run under AMOS Professional, but you will be rewarded by the fact that the Amiga's superior power over the ST can transform your programs for the better!