

File

Paul Manias

COLLABORATORS

	<i>TITLE :</i> File		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY	Paul Manias	July 26, 2024	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

1	File	1
1.1	Object: File	1
1.2	Object: File	1
1.3	Object: File	2
1.4	Object: File	2
1.5	Object: File	3
1.6	Object: File	3
1.7	Object: File	4
1.8	Object: File	5
1.9	Object: File	5
1.10	Object: File	6
1.11	Object: File	6

Chapter 1

File

1.1 Object: File

OBJECT DOCUMENTATION

Name: FILE
Version: 0.9 Beta
Date: December 1997
Author: Paul Manias
Copyright: DreamWorld Productions, 1996-1997. All rights reserved.

CHANGES VERSION 0.9B

Added: Query() to action support list.
File->Permissions

Edited: File->Comment
File->Date

1.2 Object: File

OBJECT

Name: File
Module: Files
Version: 1
Type: Simple

DESCRIPTION

The File object is used for basic file management purposes. Unlike file handles in AmigaDOS, this object allows you to find out many details on a file with almost no effort.

Files support auto-unpacking and finding capabilities. Since all File objects are tracked, there is no chance of the system allowing you to forget the closing of a File.

ACTIONS

The File object supports the following actions:

Free() Free the File object.

```

Get()    Get a File structure.
Init()   Initialise/Open a File.
Query()  Get information on a file.

```

FUNCTIONS

Some functions relating to the File object are:

```

OpenFile()  Prepare a file for reading and writing.
ReadFile()  Read data from a file.
WriteFile() Write data to a file.

```

STRUCTURE

The File structure consists of the following public fields:

```

BytePos      Current position in file.
Comment      Comment for the file, if any.
+ Date       Pointer to Time of last date stamping.
Flags        File flags and options.
Next         Next File in the chain (directories).
Permissions  RWED and other permission flags.
Prev         Previous File in the chain (directories).
Size         Total size of the file.
Source       Pointer to source of the file.

```

1.3 Object: File

FIELD

```

Name:      BytePos
Type:      LONG
Range:     Between 0 and File->Size.
On Change: Dynamic
Status:    Read/Write

```

DESCRIPTION

Specifies the current byte position in the opened file. You can change this value to read or write from different positions in a File object.

1.4 Object: File

FIELD

```

Name:      Comment
Type:      BYTE *
Inheritance: Obtained from stored file, if possible.
To Change: SetFileComment()
Status:    Read/IWrite

```

DESCRIPTION

This field points to a string that specifies the user comment for the file. Although the File object supports comments of an unlimited length, the file-system itself will have a cut-off point on the comment length. It would be unwise to assume support for anything longer than 128 bytes. If when setting a new comment you go over the limit, the extraneous characters

will be ignored.

NOTE

If you initialise an existing file and specify a comment, the old comment will be over-written with your new one.

1.5 Object: File

FIELD

Name: Date
Type: struct Time *
Inheritance: Obtained from stored file, if possible.
To Change: SetFileDate()
Status: Read/IWrite

DESCRIPTION

This field points to a Time object, which specifies the date on which this file was last stamped. This is usually considered to be the last date on which the file was changed, but this does not always have to be the case.

NOTE

If you Init() a File that already exists on disk, and if you have specified a Date, the old date will be over-written with your new one.

1.6 Object: File

FIELD

Name: Flags
Type: LONG
On Change: Cannot change dynamically.
Status: Read/IWrite.

DESCRIPTION

This field specifies the commands to use when opening the file. After opening your file you may only refer to this field for historical purposes.

Here are the flags that you can specify - they are also the same flags you can use in OpenFile().

FL_READ

This flag is the default and will open the file for reading data.

FL_WRITE

Prepares the file for writing data, starting at byte position 0. If you want to start writing from the end of the file, copy the File->Size value to File->BytePos after OpenFile() returns successfully.

FL_LOCK

Setting this will lock the file for exclusive access - no other process will be able to open the file while you are using it. If this flag is not set then the file will be open for shared access. Attempting to lock a file that is already open with shared access results in failure.

FL_FIND

This flag allows `OpenFile` to use some intelligence and try to find the file if it is not immediately located at the given `FileName`. The process involves a simple but powerful tree search.

Example

You try to load a file at `Game:Data/PIC.Crocodile`. However the `Game:` assignment does not exist. You are being run from the directory `HD1:Game/Data/` which is a directory above the required logical assignment. `OpenFile()` will find the file by using the following procedure:

```
Open - Game:Data/PIC.Crocodile (FAIL)
      Data/PIC.Crocodile      (FAIL - HD1:Game/Data/Data/PIC.Crocodile)
      PIC.Crocodile           (SUCCESS - HD1:Game/Data/PIC.Crocodile)
```

Note

The file finding feature is limited to localised searching, no attempt will be made to do a tree search of all directories in an assignment (a lengthy process).

FL_NOUNPACK/FL_NOPACK

If you specifically do not want packing/unpacking modes used on your file, specify either one of these flags. Under most circumstances you should never set these flags, because if the user does not want to use compression he/she can say so in the system preferences.

FL_NOBUFFER

Setting this flag prevents the file data from being put in the file cache. You may want to do this if it is imperative that the file physically reflects its data at all points in time.

1.7 Object: File

FIELD

Name: Next
Type: struct File *
Inheritance: Links are automatically formed when listing directories.
Status: Read Only.

DESCRIPTION

The `Next` field is generally involved with the building of directory structures. For example, if you `Activate()` a `Directory` object it will build a list of all `Files` in that directory, and point to the first `File` in the list. All further `Files` will then be joined via the `Next` field.

You can detach a `File` from the chain by using the `Detach()` action.

SEE ALSO

Field: `Prev`
Kernel: `Detach()`

1.8 Object: File

FIELD

Name: Permissions
Type: LONG
Inheritance: Init()
Default: FPF_READ|FPF_WRITE|FPF_DELETE
Status: Read/IWrite

DESCRIPTION

The permissions field gives information on the user rights for a particular file or directory. Flags currently available are:

FPF_HIDDEN

If the file should be hidden from this user then this flag will be set.

FPF_DELETE

If the user has delete rights for this file then this flag will be set.

FPF_EXECUTE

Execute rights can only be given to executable files and scripts.

FPF_PASSWORD

This is a special flag that can lock a user out from a file until the correct password is given. Note that support for this flag is still under-way and currently most file-systems are not able to support it directly.

FPF_READ

This flag allows the user to read the file so that its contents may be viewed. Note that read access does not give execute access.

FPF_SCRIPT

Standard text files which are set with this flag will be treated as scripts, which are capable of executing multiple commands. You will need to set this flag in conjunction with EXECUTE in order to run the script.

FPF_WRITE

This flag allows the user to write data to the file. While this does not give delete access, be warned that it is possible to clear a file to a size of 0 bytes.

1.9 Object: File

FIELD

Name: Prev
Type: struct File *
Inheritance: Links are automatically formed when listing directories.
Status: Read Only.

DESCRIPTION

The Prev field is generally involved with the building of directory structures. For example, if you Activate() a Directory object it will build a list of all Files in that Directory, and point to the first File in

the list. The Prev field will assist in moving backwards through the chain.

You can detach a File from the chain by using the Detach() action.

SEE ALSO

Field: Next

Kernel: Detach()

1.10 Object: File

FIELD

Name: Size

Type: LONG

Inheritance: Init()

Status: Read Only.

DESCRIPTION

Specifies the current byte size of the opened file. This field can only be changed if a call to WriteFile() requires that the file size is expanded. It cannot be dynamically changed and cannot be shrunk in size.

1.11 Object: File

FIELD

Name: Source

Type: APTR

On Change: Cannot change dynamically.

Status: Read/IWrite.

DESCRIPTION

Points to a source object, which may be either a FileName, MemPtr or recognised system object. The source will be used to read data from or write data to. Note that FileName and MemPtr are fully supported by the system at all times, but not all objects can be expected to support read/write actions.
