Examine1.4

Neil Carter

COLLABORATORS			
	TITLE : Examine1.4		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Neil Carter	February 12, 2023	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Examine1.4

1.1 main

```
                                         Examine V1.4

                    Written by Neil Carter

                        PUBLIC DOMAIN
```

1.2 introduction

```
             INTRODUCTION
```

```
Examine V1.4 is a little program I wrote to help me write scripts and
identify unknown files.  It is quite small, and very versatile (its format
```

option is compatible with List).

Features:

:-) Uses, but does not absolutely require,
FileID.library
. With it, you
can easily identify most Amiga filetypes. Without it, Examine will just tell you whether the file is binary or various forms of ASCII text. FileID won't be invoked unless a function (or format token) which requires it is called, saving time and memory.

:-) Flexible format command. Most format tokens are exactly the same as the standard List command, so if you know that, you know Examine. Extra tokens allow you display various FileID values or do little ASCII or hex dumps. If desired, you can choose a different format for directories.

:-) Also capable of performing certain handy script functions. For example, It can return WARN if the filename is that of a directory, or if it's an executable. It can return the FileID type as a return code, allowing you to quickly confirm filetypes in your scripts.

:-D It's pretty good actually! I use it all the time. Source in Amiga E is included, so you can have a good laugh at it.

1.3 requirements

REQUIREMENTS

Examine V1.4 requires:

- Kickstart 2.04 or better on some kind of Amiga.
- FileID.library version 2 or higher if you want file identification. Version 7 can identify about 700 different types of file, including a wide range of compressed data formats and music modules. A copy of the version 7 FileID.library is included, without any supporting documentation or extra catalogs. If you want to download the full archive, it can be found on Aminet under util/libs/fidlib70.lha.
- A little AmigaDOS experience.

1.4 basicusage

BASIC USAGE

The command line template is:

```
     FILENAME,BRIEF/S,FULL/S,Q=QUICK/S,NOFILEID/S,FORMAT/K,DFORMAT/K,
     NUMBYTES/K/N,LIST/S,ALL/S,VERSION/S,ID/S,RETURNID/S,EXE=EXECUTABLE/S,
     DIR=DIRECTORY/S,WINDOW/S
```

The simplest way to use Examine is thus:

```
     Examine <filename>
```

This returns:

```
     "Filename:           <filename>
      Path:               <path>
      Size in bytes:      <length>
      Dir entry type:     <type - file or dir>
      FileID name:        <fileid type name>
             code:        <fileid code number>
             class:       <fileid global file class text>"
```

Parameters and options:

FILENAME   This gives a filename/filepath to the object you want to
               examine. It can be a file or a directory. At the moment,
               directory scanning and filename patterns are not supported.

BRIEF/S   Gives the information in the format:

```
                    "Object <filename> is a <fileid type name>"
```

FULL/S   Gives the information in the format:

```
                    "Filename:           <filename>
                     Path:               <path>
                     Comment:            <comment>
                     Datestamp:          <time> <day>, <date>
                     Size in bytes:      <length>
                     Size in blocks:     <blocks>
                     Protection:         <protection bits>
                     Dir entry type:     <type - file or dir>
                     Begins with:        <hex dump> <ascii dump>
                     FileID name:        <fileid type name>
                            code:        <fileid code number>
                            class:       <fileid global file class text>"
```

Q=QUICK/S   Just returns the FileID type string.

NOFILEID/S   Stops FileID from being opened. This is only necessary if
               you wish to see whether a file is text or binary, as Examine
               only opens FileID if it's available and if it's called for.

FORMAT/K   Allows you to define a List-style format string.
               See
                Defining Custom Formats
                 for details.

DFORMAT/K   Allows you to define a separate format string for use with
               directories only (where entries such as "file size" have no
               meaning). If you don't specify it, FORMAT will be used
               instead.

```
NUMBYTES/K/N   Specifies the number of bytes to be output by the %h and %o
               tokens.  The default is 8 bytes.

LIST/S         Not implemented.  When it is, it will allow Examine to scan
               directories in a similar way to list, identifying files as
               it goes.

ALL/S          Not implemented.  When it is, it will cause recursion into
               subdirectories when in LIST mode.

VERSION/S      Not implemented.  If I ever do implement it, :-) it will
               do a Version-style version string.  It's the same as the %v
               token.  That isn't implemented, either.

ID/S           Returns only the FileID code number.

RETURNID/S     When this switch is set, the return code ($RC) set by
               Examine will be the FileID code number.  This can be used
               in scripts to identify filetypes before processing files.

EXE=           When this switch is set, Examine will return WARN ($RC=5)
EXECUTABLE/S   if the file is an executable.  This does not require
               FileID.library.

DIR=           The same as the above, except it returns WARN if the
DIRECTORY/S    filename refers to a directory.

WINDOW/S       Causes Examine to display its output in a Requester instead
               of in the shell.  It's not very good if (like me) you have
               a proportional screen font, as it's hard to tabulate the
               data.
```

1.5 definingcustomformats

```
DEFINING CUSTOM FORMATS

These are the currently supported format tokens:

* -------------Also present in List.
 # ------------Causes FileID to be loaded.
  + -----------Unused in this version.

*    %a        Protection bits. These are shown as "hsparwed", where:

                    d    Delete enabled
                    e    Executable
                    w    Write enabled
                    r    Read enabled
                    a    Archived
                    p    Pure (unlike this program :-)
                    s    Script
                    h    Unused - supposedly "hidden"

               Bits are shown as the letter above if they are on, or as a
```

		dash or they're off.
*	%b	Size in blocks.
*	%c	Comment. If there isn't one, "none" is returned.
*	%d	The file's date. Just the date, not the time or weekday.
	%e	Directory entry type (ie. file or directory)
*	%f	Full (absolute) path
#	%g	FileID global file class bits (sfgmiepx). The bits mean: x Executable p Packed (PowerPacked, for example) e Encrypted (might be password protected) i IFF (any kind of IFF file) m Music (can be unreliable!) g Graphic image f Formatted text s Script Read the documentation on FileID to find out exactly what these terms mean. They're quite broad terms, and often take in things you wouldn't expect!
	%h	First few bytes hex dump. The number of bytes depends on the NUMBYTES/K/N keyword.
#	%i	FileID code number.
*	%k	Disk key.
*	%l	Size in bytes.
+	%m	Unused.
*	%n	Filename, as specified by the user.
	%o	First few bytes text dump. Again, the NUMBYTES/K/N keyword controls the number of bytes displayed.
*	%p	Path as supplied.
+	%q	Unused.
#	%r	FileID global file class text. This token will return a string in the following format: (Script)(Text)(Graphics)(Music)(IFF)(Encrypted)... (Packed)(Executable) Obviously, only the relevant words will appear. If the file type has no class definition, the word (None) will appear.
#	%s	FileID type name string.

```
*    %t        Time, taken from the file’s datestamp.

  +  %u        Unused.

  +  %v        Not implemented.  When (if) it is, it will scan the file for
               a version string beginning with "$VER:" and will return the
               version number (just the number, probably).

*    %w        The weekday from the file’s datestamp.

  +  %x        Unused.

  +  %y        Unused.

 #   %z        This token returns either "a" or "an", depending on the
               first letter of the %s token.  This is present purely for
               fussy gits like me who don’t like seeing strings such as:

                    "Object Bobbins.32C" is a IFF picture/brush"
                                           ^^^
               Use this instead:

                    "Object %n is %z %s"

     %%        Just prints a real "%" sign.  In case you want one.  :-)

     *N        Inserts a linefeed into your format text.  This is a
               standard AmigaDOS token, so you can use it in other
               programs.

     *"        Inserts a double quote into your format text.  Ditto.
```

The tokens are case sensitive.

Note that it doesn’t make any sense to use certain tokens when the examined object is a directory. Tokens such as %l (length in bytes) will just return "N/A" (not applicable). If this isn’t appropriate, you can use the DFORMAT/K keyword to specify a different format for directories.

You can also use the paragraph sign "¶" instead of "%" if you want (on my keyboard, it’s <alt-p>). This is an attempt to avoid clashes on the command line when trying to pipe its output into another command which uses similar tokens, with the backtick "`" symbol.

Incidentally, I notice that there’s a new official version of List floating around Aminet, which has extra tokens for dealing with the protection bits etc. of multi-user filesystems. Naturally, those bits clash with mine, :-(so I might change them to capital letters sometime.

1.6 scriptfileapplication

SCRIPT FILE APPLICATION

How to identify an LHA archive before unarchiving it:

```
     .key FILE/A
     .bra {
     .ket }
     ;Setting parameter brackets to "{}" is just a personal perversion.
     ;Don't forget that the ".key" instruction must be on the first line!

     ;Here, Examine is run in BRIEF mode so it puts just the FileID string
     ;in a global ENV variable.  It also uses RETURNID to set the return
     ;code to the FileID code number.  That number for an LHA archive is
     ;"71", so we can check for that.

     Examine >ENV:ExLhaMessage{$$} "{FILE}" BRIEF RETURNID

     ;Check that the file is an LHA archive.

     If $RC EQ 71
          ;File IS an LHA archive, so we process it.
          LHA x "{FILE}"
     Else
          ;File was something else.  Complain to the user!
          Echo "File was a $ExLhaMessage"
     EndIf

     ;Clean up....
     UnSetEnv ExLhaMessage{$$}
```

How to check if a filename is a directory:

```
     .key FILE/A
     .bra {
     .ket }

     ;The directory keyword causes Examine to return WARN if the filename
     ;refers to a directory.  Examine will try to display its usual
     ;output, so you should redirect it to NIL:.

     Examine >NIL: "{FILE}" DIRECTORY

     If WARN
          Echo "It's a directory"
     Else
          Echo "It's a file"
     EndIf
```

How to return an appropriate string depending on the filetype:

```
     Examine >ENV:ExamineMessage{$$} FORMAT "File type *"%s*" (%l bytes)"
          ... DFORMAT "Directory"
```

Returns:

```
     File type "unknown executable file" (39206 bytes)

     ...or...

     Directory
```

This is useful, as directories obviously have no file size. If you were to call the token "%l" on a directory, you would get the string "N/A" instead.

1.7 history

HISTORY

```
Version 1.0     Basically a rip-off of the original FileID.library example
                code.  It didn’t really do anything special, but it was the
                first serious program I wrote in Amiga E, so I was quite
                proud of it!

Version 1.1,    I’m not sure what happened to these versions! :-)  I suffer
        1.2     from "version number bumping syndrome", so these versions
                probably evolved into V1.3.

Version 1.3     Supported output in various different formats.  The code was
                generally tidied up and some simple (potential) bugs were
                removed.

Version 1.4     Completely re-wrote the engine of the program.  As I was
                adding custom output formatting, I though I might as well
                strip out the original display routines and just do
                everything through the format parsing routine.  Much
                simpler!  In addition to its own functions, Examine can
                now do most things that List can do, except for
                directory scanning.
```

1.8 todo

TO DO...

I would like to add the following features at some point, but please bear with me; I’m a very slow programmer.

- Directory scanning and recursion via the LIST/S and ALL/S switches. This will require a fairly hefty redesign of the program’s inner workings. I don’t like the thought of it, but since this is the one function which would be most useful to me, I probably will do it.

- Implement the VERSION/S switch and the %v token. I don’t really want to do it, so unless you, the dedicated user, demand it of me, it will most likely be removed!

- Add a means for tabulating data. Something like "%n[20]", for example, allowing the following kind of format:

  ```
  "%n[20]%l[10] %s"
  ```

  ```
  ...returns...
  ```

```
"s:startup-sequence       1227 pure ASCII text file"
```

This is essential if I allow directory scanning (otherwise the output'll be all over the place!), so this seems likely to be added.

· Maybe add my own requester so I can have tabulated data when the WINDOW/S switch is used. It looks really messy at the moment.

· Hmm... dunno. Suggestions are encouraged!

1.9 bugs

BUGS

There are some. :-)

Firstly, Examine is currently not pure. As such you should probably not make it resident - since it's supposed to be used in scripts, the chances of something causing it to be executed twice simultaneously are pretty high. It will probably crash if this happens. I'm not quite sure why, though. This'll get fixed when I add directory scanning.

Secondly, very rarely, I've seen it crash for no reason at all. It's difficult to point the finger at anything in particular, as these crashes could just as easily be caused indirectly by something else I'm testing while it's running. It could equally be FileID itself. Such crashes are extremely rare, though.

I don't have an MMU, so I can't test for Enforcer or Mungwall hits. Any volunteers? ;-)

1.10 author

AUTHOR

Hi.

I'm Neil Carter, a freelance computer graphic artist and sometime computer programmer, living in the nethers of London. I've just graduated from a three year Interior Design BA at Kingston University, and I'm searching for computer modelling and visualisation work relating to architecture.

I'm currently working on several programming projects in my free time, including a ClassAction type program using FileID, and an enormous space combat strategy game thing which I might finish somewhere around the end of the century at this rate. I write in Amiga E, 68000 assembly and occasionally AMOS.

My machine is an old revision 6 A500, with a Kickstart 2.05 (!) ROM. It has 5 megs of RAM, a 50 meg hard drive ;-) and two floppy drives. Well, I like it! :-P

I can be contacted at:

```
     n.carter@kingston.ac.uk       (Up until about June/July 1997, but
                                   maybe longer.)

     Neil Carter                   (Parents' address - be nice!)
     1 Langer Close
     Lincoln
     LN6 0SX
     England
```

Bug reports, comments, etc. are welcome. Flames >NIL: please!

1.11 credits

CREDITS

Thanks a lot to Bloodstone of Syndicate, the author of FileID.library.
Nice work!

1.12 disclaimer

DISCLAIMER

You use this program entirely at your own risk. If it blows up on you, I
will not accept any kind of responsibility.

That said, I trust the thing enough to use it every day in several vital
scripts on my Amiga. However, you should read the
Bugs
section.