LeoUtils

Henrik Herranen and leopold@cs.tut.fi

COLLABORATORS			
	TITLE : LeoUtils		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Henrik Herranen and leopold@cs.tut.fi	August 24, 2022	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

LeoUtils

1.1 LeoUtilities - Table of contents

The programs marked as ANSI are written in ANSI C and can be compiled on any system that have proper file names and an ANSI C compliant compiler.

1.2 Disclaimer

Disclaimer

LeoUtilities have been carefully tested by me. I have been running these programs through Enforcer and MungWall and I have great confidence in that the programs run properly in almost all conditions. At least I have not been able to crash them. Yet:

The author cannot be held liable for the suitability or accuracy of this software and/or this manual. Any damage directly or indirectly caused by the use of this software is the sole responsibility of the user.

In plain English: In the very (!) unlikely case that your HD crashes, your monitor blows up and your girlfriend decides she'll never want to see you again while you are playing with LeoUtilities and you think these utilities are the cause of it, I'll in no case buy you new ones (not even a girlfriend).

1.3 Copy right, not wrong!

Copyright

LeoUtilities are Freeware. The author will keep all his legal rights to this software package with the following exceptions:

You are allowed to make a copy of this program if you are able to meet the following simple rules:

1. You may not modify the software binaries nor the fonts. You may not modify the source files for other reasons than fixing bugs or creating patches to make them compile on your favourite compiler. You may only pass the original package forward, not a version you have modified.

2. You may not charge more than for the media of a disk containing LeoUtilities. And remember that a double density or even a high density floppy disk does not cost $4 or 30 FIM. A 20 MB floptical just might.

3. If you use some or all programs of this package, you are strongly encouraged to send a postcard to the author .

A written permission from the author and an appropriate fee is required before this package or any part of it may be included on a magazine coverdisk, a CD-ROM or some other kind of commercial product.

Remember that this program is Freeware. It means that you don't have to pay for this. Anyhow, the longer you live from Finland, the more I'd like to see a postcard from you. Which, of course, is not to say I wouldn't like to have postcards even from friends of mine :-) .

1.4 Introduction

Introduction

LeoUtilities is a collection of some small utilities made by Leopold-Soft to make everyone's life a bit easier and more fun.

All Amiga binaries are startable and usable only from the Shell, with the exception of MemTest which can also be started from the Workbench. They require at least OS 1.3 to work and some require OS 2.0 or/and a 68020 CPU. All programs are compiled to work with a standard 4K stack. All binaries are compiled with the SAS C developement system (ver. 6.51).

Some programs are written in standard ANSI C and can be compiled on any system that has an ANSI C compliant compiler. Some programs may need significant amounts of stack.

1.5 Author - Leopold-Soft

Leopold-Soft

If you want to send me a postcard or just chat, send your mail to the following address:

Henrik Herranen

Opiskelijankatu 38 A 7

33720 Tampere

Finland

e-mail: leopold@cs.tut.fi

My WWW homepage may be found at URL "http://www.cs.tut.fi/~leopold/". There you may, if you like, take a look at my cats, read about Pink Floyd or follow the outlines of Babylon 5. Also the newest version of this piece of software among with my other PD/SW products will be kept there.

PS. When you create AmigaGuide documents, please don't write them with a fixed line length. Use the WORDWRAP keyword like I've done here, ok?

PS2. Bug reports and enhancement suggestions are welcome.

1.6 The history of LeoUtilities

The history of LeoUtils

An asterisk after the LeoUtils version number indicates that version was put on public release.

LeoUtils 1.0* 1.1 1.2* 1.3* 1.4* 1.41 1.5*

Launch date 940425 940505 940519 950211 950404 960313 961125

LeoClock 1.01 1.01 1.01 1.10 1.10 1.11 1.11

LeoTime 1.0 1.01 1.01 1.01 1.01 1.01 1.01

FakeCpu 1.0 1.0 1.0 1.0 1.0 1.0 1.0

LeoTune 2.1 3.0 4.1 4.13 4.13 4.13 4.14

GuideCat 1.0 1.1 1.1 1.1 1.1 1.1 1.11

Words - 1.0 1.0 2.1 2.1 2.1 2.1

LeoLib - - 1.0 1.0 1.0 1.0 1.0

MemTest - - - 1.1 1.1 1.1 1.1

Wc - - - 1.0 1.0 1.0 1.0

Beep - - - - 1.0 1.0 1.0

ScrDump - - - - 1.01 1.01 1.1

Subtitle - - - - - - 1.0

Leopold.font 1.0 1.0 1.0 1.0 1.1 1.1 1.1

1.5 Subtitle added; GuideCat updated.

1.41 A new, 4 byts shortened version of LeoClock , modified Makefile.

1.4 Two new progrmas added, Beep and ScrDump .

1.3 LeoTune enhanced again and a new program, MemTest is included.

Size of LeoClock reduced.

1.2 Major new features to LeoTune gives justice to a new version

number. LeoLib introduced.

1.1 This version exists only because I mailed LeoUtils at this

stage to a friend of mine, before I noticed the random number

generator of LeoTune wasn't particularly good.

Words is a new program.

1.0 Initial release of LeoUtils containing LeoClock , LeoTime ,

FakeCPU , LeoTune , GuideCat and the wonderful Leopold.font .

1.7 LeoClock - The smallest Workbench clock

LeoClock

1996-03-26

LeoClock is by far the tiniest clock that I know of that really works under OS 2.0.

Why use LeoClock?

+ Uses less than one disk block

+ Starts to the default Workbench or public screen.

+ Locates itself automatically up to the workbench border.

+ Opens it's window to the same height as the WB border.

+ Boosts its priority up to 4 to allow smooth operation. Returns the
original priority upon exiting.

+ Works well under OS >= 2.0.

+ No Enforcer hits.

+ Eats as little processor time as possible = only 1 loop / s.

+ Executes a WindowToFront() every full minute -> keeps visible.

+ If you are using a font with proportional numbers (which are very
rare, but I happen to use one) and LeoClock notices that its new
numbers won't fit in the window, an automatic window resizing &
locating is done (check this out with LeoProp.font 11).

+ Creates a smaller window if the time is less than 10 hours.

+ Despite LeoClock may change its window, the right edge of the window
never moves by itself.

+ Uses only some 20 bytes of stack (+ ~2k that the OS routines need).

Why dislike LeoClock?

- Not configurable in any ways.

- No analog mode.

- Looking at the source code may cause headache and temporary loss of
mind control.

- Although technically working, butt-ugly under OS 1.3.

- Not startable from Workbench.

Summa summarum, kipikapi to add these lines to your S:User-Startup:

Stack 2048

Run <>NIL: LeoClock

Stack 4096 ; Or whatever you want

Credits

Thanks to Pasi Ojala (albert@cs.tut.fi, http://www.cs.tut.fi/~albert/), Janne Salmijärvi (jhsa@cs.tut.fi, http://www.cs.tut.fi/~jhsa/) and Timo Kaikumaa (timok@cs.tut.fi, http://www.cs.tut.fi/~timok/) for their ideas for making the code smaller for v1.10 and v1.11 (it was not my idea to get rid of two library names out of three).

Program history

1.11

960326 440 Timo Kaikumaa ate 12 bytes out of it.

1.10

950210 452 Optimizimizing... The last 4 bytes required 8 hours of

work!

950209 468 Abandoned LeoClockLocation2, optimized code with the

help of Pasi Ojala and Janne Salmijärvi.

1.01

930628 488 Made an extra version, LeoClockLocation2, which opens

its window in a slighly different location.

Chopped some extra bytes and got the executable

squeezed to exactly one OFS block. 488 bytes, that is.

I strongly believe LeoClock can't be size optimized

any more without loss of functionality.

1.0

930427 504 Two days of contiguous crunching: 504 bytes. The

source code has become quite well obfuscated.

930425 1136 First working Assembler version: 1136 bytes. Now for

some optimizing...

930424 4104 C version up and running. Compiled size: 4 KB with

Dice. Not bad for a C compiler.

1.8 LeoTime - A program execution time counter

LeoTime

1994-04-25

LeoTime is another one-disk-blocker. It allows you to measure how much time it takes for Amiga OS programs to run. It measures the real time the process uses and not processor time, since that information is at least not easily available under Amiga OS (if at all).

You may test LeoTime by typing something like leotime list.

If you want to redirect the output of the command you wish to run, you may do this by putting a pair of quote characters around the command, like: leotime "list sys: all nohead >ram:MyHardDrive". I chose to implement it this way to allow you to redirect either the output of LeoTime or the command to be run.

LeoTime requires at least a 68020 processor and OS 2.0 to run. If tried to run on a lesser processor, it will print an error message and gracefully exit (you may test this with FakeCPU if you want to (see how useful these utilities are :-) ?). If run under OS 1.x LeoTime will simply exit. It can't display an error message because it uses OS 2's printing routines. I could have written this program to run on the 68000 and OS 1.x, but I didn't, because I don't like writing assembler even a line more than is absolutely necessary.

By the way, the only reason why this program is called LeoTime and not just Time is that Time is one of the standard preferences programs of the Workbench.

History

1.01

940425 452 Now allows correct redirection if the command to be

run is surrounded with quote characters.

1.0

940418 512 First version, but redirection doesn't work like it

should :-(.

1.9 FakeCPU - See and lie to the OS about your CPU and FPU types

FakeCPU

1994-03-18

FakeCPU lets you see what kind of a processor and math coprocessor the OS thinks you have.

You may also change the processor type the OS thinks you have by specifying the processor type you want on the command line:

Usage: FakeCPU [680x0 (x=0..4) NOFPU 68881 68882 FPU040 FPU040+882]

The difference between FPU040 and FPU040+882 is that the latter is the 68040 with the 68882 emulating software.

History

1.0

940418 3432 First working version, uses Amiga OS's Printf instead

of C's stdlib's printf.

1.10 LeoTune - Introduction to the fortune cookie program

LeoTune

1995-02-10

LeoTune is a fortune cookie program. The idea is that every time you invoke LeoTune, it randomly gives you a short thought from it's database. Although LeoTune is downwards compatible with the most common fortune file format, it can do much more.

The fortune cookie database is a simple ascii file where you may add your own short thoughts or funny stories.

Installation on an Amiga System

Installation on a Unix System

Specifying the fortune cookie file

Creating a fortune cookie file

Creating the fortune index files

Complete command option list

Word wrapped fortune commands

Program history

This package does include the source of the program as well as an Amiga compatible binary file. You may compile your own version on any computer system if you have an ANSI C compliant compiler available. However, if the character set your OS uses is not compatible with the ANSI / ISO LATIN 1 character set (has multiple EOL characters like MS-DOS & VMS), LeoTune will probably not work.

1.11 LeoTune - Installation on an Amiga System

Installation on an Amiga system

Move the LeoTune executable file somewhere on your default path.

1.12 LeoTune - Installaton on a Unix System

Installation on a Unix system

Compile LeoTune. With gcc it is done like this:

51% gcc -ansi LeoTune.c -o leotune

52% strip leotune

You may also try to compile LeoTune simply by saying make leotune. I have included a standard Unix Makefile with this package.

1.13 LeoTune - Specifying the fortune cookie file

Specifying the fortune cookie file

If you just want to quickly test LeoTune (and you have compiled it if you are on a non-Amiga system (see Installation on a Unix system), type the following lines:

leotune -i LeoTunes

leotune LeoTunes

As a default, LeoTune tries to read its fortune file from the file called Fortunes:LeoTunes. This is a very Amiga specific name, but no worry, Unix (and multi-user Amiga system) users:

If LeoTune finds an environmental variable called HOME, it tries to read the fortune file from $(HOME)/text/LeoTunes.

If you don't like these defaults, you may define an environmental variable called LEOTUNEDIR, and LeoTune will try to read its file from $(LEOTUNEDIR)/LeoTunes. You may add a trailing "/" to the directory name (or a ":" on the Amiga), but it is not required.

If you still aren't satisfied, you may define the fortune file and its path as defined later in this document as a command line option for LeoTune (like we did in the example above).

1.14 LeoTune - Creating a fortune cookie file

Creating a fortune cookie file

The fortune cookie file is of a very simple format: 8-bit ISO 8859-1 (Latin 1) as used in Amiga OS, newer Unix systems and MS Windows. Two fortunes are separated by a line with only '%%' or '##' on it. For instance, below is a simple fortune file with three fortune cookies.

This is the first fortune.

%%

And this is the second one.

And way longer it is than was the first one.

##

Yoda the Jedi-master always strangely backwards talked but never any

spelling errors made. Selective he was in wrongly speaking.

%%

You probably want to name your fortune database file LeoTunes, and put it in a directory you specified earlier (look at Specifying the fortune database file).

I have included a small example fortune cookie file in this directory surprisingly called LeoTunes.

About the Fortune Ending Characters

As you have already noticed, a fortune can end in two different ways. Why?

The '%%' ending is the traditional and universally adopted standard separator of fortunes. LeoTune treats fortunes like this as already formatted, and it will simply display them just as they were in the fortune file.

However, the new '##' ending is another matter. By ending your fortune with this string you tell LeoTune it can handle the word wrapping for you. So, you may write as long lines you wish, and LeoTune will format them for you. You may also tell LeoTune to display the fortunes with different widths depending on your terminal (see Command options). With word wrapped fortunes you may also use several word wrapped fortune commands . I strongly encourage the use of this new format I just invented.

1.15 LeoTune - Creating the fortune index files

Creating the fortune index files

Before you may ask LeoTune to show one of the fortunes, you must ask it to create two index files. This is done by typing leotune -i. LeoTune will announce how many fortunes it found and inserted to the index files. It will also tell you how many new indexes were created.

After this you are ready to go: just type leotune and have fun. And, remember, every time you touch the fortune cookie file, you must run LeoTune with the -i option.

1.16 LeoTune - A complete command option list

A complete command option list

Every command option has a synonym to make LeoTune compatible with both Unix and Amiga command option naming conventions.

-h Shows a short credit and help page with all the commands and

? their meanings listed.

-i Creates the two LeoTune fortune index files. The first one

index includes all the fortune cookies, and the second one only

those with mostly 4 lines of not more than 79 characters each.

The latter index file is mainly meant to be used to autocreate

short Unix mail signatures.

-s Normally LeoTune shows one blank line before and after the

short fortune cookie. This option inhibits that.

-4 Shows a "signature" fortune with mostly 4 lines of not more

sig than 79 characters each. This option will force the width of

word wrapped fortunes to 79 columns. This option is meant to

be used to create fortunes that follows the official signature

netiquette as used in the UseNet community. To create a UseNet

signature you probably also want to use the -s option.

-7 Convert the following characters: "éüåäöÉÜÅÄÖ" to their 7-bit

iso11 Iso 11 equivalents: "`~}{|@^][\". All the other 8-bit

characters are converted to question marks. This is a very

Finnish option and may not be of much use elsewhere.

-t x Shows a specific fortune index x where x must be an integer

tune x between 1 and the amount of fortunes in the fortune file. If

-4 is also defined, the fortune will be taken from the

signature fortune database.

-g x Shows all fortunes that include the specified string x. This

grep x option will always turn the -4 option off and the -s option

on. The string x is case-insensitive.

-G x Like -g / grep, but case-sensitive. So, Leopold will be a

GREP x different thing from leopold or LEOPOLD.

-w x Defines the maximum width of word-wrapped fortunes. The default

width x is 79 or whatever you have set in your environmental variable

$COLUMNS. This option will have no effect if the -4 option is

used.

filenam If you provide a filename, that file will be used as the

fortune cookie base file instead of the default file.

1.17 LeoTune - Word wrapped fortune commands

Word wrapped fortune commands

A fortune that ends with a line that consists only of '##' is considered to be a word wrapped fortune by LeoTune, i.e. LeoTune will take care of word wrapping instead of you.

Newline

A Newline character is the equivalent to a whitespace. However, any n number of successive Newlines (where n > 2) is considered as n-1 Newlines.

Escape Commands

The backslash character '\' is considered as a command escape character and always begins a command. If the character(s) after the escape character can not be recognized as a command, the escape character is simply disregarded.

\> Tells LeoTune to indent every line to the column the cursor is

currently on until the next Newline command is encountered.

The indent commands do not nest.

\< Tells LeoTune to unindent the text indented with \>.

\\ Displays a backslash character '\'.

An Example of Using Word Wrap Commands

So, let's say you have the following fortune in your fortune file (the quoted lines begin with '-> '):

-> Hofstadterin laki:

->

-> -- \>Projektin valmistumien vie aina enemmän aikaa kuin olit

-> arvioinut. Tämä pätee myös silloin, kun olet ottanut

-> Hofstadterin lain huomioon.

->

-> That was Finnish, you know :-)

-> ##

If you now invoke LeoTune first with the option -i and then with the option -w 40 , you'll get a following fortune displayed:

-> Hofstadterin laki:

-> -- Projektin valmistumien vie aina

-> enemmän aikaa kuin olit arvioinut.

-> Tämä pätee myös silloin, kun olet

-> ottanut Hofstadterin lain huomioon.

-> That was Finnish, you know :-)

Suggestions?

This concept of using control characters in a fortune file is all new. If you have suggestions of important new control characters you'd like LeoTune to handle, please let me know. I'll do what I can.

1.18 LeoTune - History

History

4.14

961119 17612 Minor changes to source to get Leotune to compile

without warnings with the newest version of SAS-C.

4.13

950210 17348 Minor bug fix: Non-set system clock would break

LeoTune on a non-Amiga system. Now gives a warning

message & resets to fortune #0.

4.12

941016 17276 A minor bug in automatic indenting "\>" command

fixed.

4.11

940613 17244 Grep options -g and -G sped up by a factor of 2 - 3.

The speed is now almost independent of the length of

the string to be searched for.

Added a 16 K disk buffer for faster reading. Didn't

affect the speed on my Amiga with the SAS C compiler.

4.1

940519 16884 Minor source code cleanup. LeoAnsi.h replaced with

LeoLib .

940517 16884 Cleanup of word wrap parsing code.

940516 17156 First word wrap commands ’>’ and ’<’. Source code is getting quite cluttered.

4.0

940515 16728 Added new word wrap capabilities and due to popular demand a case-sensitive version of grep (-G).

3.1

940511 15164 Rewrote (again) the random number generator. If compiled on a non-Amiga system, the randon number simply is the time in seconds modulo the amount of fortunes. On an Amiga system the random number is the amount of clock ticks (50/sec) shifted right one time modulo the amount of fortunes. This seems like a good solution unless you take several fortunes in a row on a non-Amiga system.

3.0

940505 15468 Replaced toupper() with my own table upcase[] from file LeoAnsi.h -> Grep works now correctly with 8-bit characters even if you have a braindead C compiler.

940427 15212 Added a somewhat inefficient -g (grep) option. Tried to compile with DICE -> 13916 bytes. However, the DICE version doesn’t work with 8-bit characters, and it’s 20% slower anyway, so who cares about a lousy 10368 bits :-) ? I don’t, because this is no LeoClock .

940426 13944 Added the -t (specific fortune number) option. Replaced the rand() function with a pseudo-random one: the time in seconds bit-reversed.

2.1

940414 13452 Discarded all #ifdefs in compilation for Amiga / Unix. Much cleaner now, and the environmental variables can be used on the Amiga, too (great for multi-user systems).

Default file names shortened -> usable with older Unixes with a maximum of 14 characters / file name.

940413 12964 Switched to SAS C compiler on the Amiga. Provides better ANSI C compatibility checks than DICE which compiles anything remotely like C code. Unluckily the binary file size growed significantly.

2.01

931012 10820 Corrected some bugs.

2.0

931011 10812 Lots of new features added, like special signature

mode and 7-bit scandinavian character conversion.

Conversion is done correctly only if the original file

uses the standard ANSI character set used in Amiga OS,

newer Unix systems and MS Windows.

1.1

930920 Managed to compile under Unix. Several non-ansisms in

C-code corrected.

1.0

920907 First working version of the program.

1.19 GuideCat - Types AmigaGuide documents

GuideCat

1996-11-19

GuideCat converts an AmigaGuide formatted hypertext file into plain Ascii with or without ANSI compatible text formatting control escape sequences. This will allow non-Amiga owners to read AmigaGuide files, like this one. I've seen some converters before, but they didn't do a particularly good job and the one I tried the last time simply choked on long lines. This behaviour really is inexcusable nowadays when AmigaGuide format supports word wrapping.

Features:

+ standard ANSI C - can be compiled anywhere where an ANSI C compliant

compiler is available

+ handles long lines (works on a word rather than line basis)

+ handles wordwraps if so defined in the AmigaGuide file

+ understands underline, bold and italics

+ understands the escape character \

+ can create magic numbers to implicitly identify links and nodes

- not particularly fast (34 sec on my 14 MHz 68020 Amiga for a 200 K

file), but usually fast enough.

- no Amiga pattern matching (Unix users couldn't care less :-))

- doesn't invoke links from other files

Installation on an Amiga System

Move the GuideCat executable file somewhere on your default path.

Installation on a Unix System

Compile GuideCat. With gcc it is done like this:

51% gcc -ansi GuideCat.c -o guidecat

52% strip guidecat

You may also try to compile GuideCat simply by saying make guidecat. I have included a standard Unix Makefile with this package.

Command Line Options

-h or ? Shows the help page.

def: 0 1 2

-nI or -ni Turns Italics on/off in node headers on off off

-nB or -nb Turns Bold on/off in node headers on on off

-nR or -nr Turns Reverse on/off in node headers on off off

-nC x or -nc Sets color on/off in node headers off 3 off

-lI or -li Turns Italics on/off in links off off off

-lB or -lb Turns Bold on/off in links off on off

-lR or -lr Turns Reverse on/off in links on off off

-lC x or -lc Sets color on/off in links off 3 off

-cR or -cr Turns Reverse on/off for colors off off off

-cC x or -cc Sets colors on+hilite/off for colors 2 2 off

-I or -i Switches node index numbering on/off on on on

-S or -s Switches stupid terminal mode on/off on off on

-B or -b Brackets around nodes+links on/off off off on

-P pr -p plain output on/off off off on

-w x (x>=20) Set display width if WORDWRAP used 79 79 79

-d0 .. -d2 Use defaults 0..2

filename the named file will be read as an AmigaGuide file.

Index numbering (-I) allows you to get an explicit index number for every node in the document. The index number is mentioned also every time a link to that node is made. The numbering starts from 1. Since GuideCat does only a one-pass scan, it will allocate new numbers as they are mentioned for the first time.

Some terminals can switch various modes like Bold or Reverse on, but can't turn them off one-by-one. If the stupid terminal option (-S) is selected, GuideCat uses a different approach to turning these special effects off: It turns them all off and then restores the modes that should be restored.

The brackets option (-B) is most usable in conjunction with the plain output option (-p).

The plain output (-P) option will inhibit all ANSI code formatting. This will give you some kind of a result with a dumb terminal, although I must admit, that reading a hypertext document without any clues at all of the links and nodes might be a bit painful. This option is automatically turned off if you define any option that changes the text style (-lX, -nX, -cX).

The defaults are meant to be used as follows:

-d0: Default defaults, colors in the AmigaGuide file will be shown

-d1: Emphasize node headers and links as color

-d2: For dumb terminals, no formatting at all

Command line options can be combined in any order. Only the files mentioned after the options are affected. So, if you say something like guidecat pink -p floyd, only the file floyd will have no formatting.

If the WORDWRAP command line option is defined in the AmigaGuide document, GuideCat will make all lines fit into the given amount of columns. The default is 78, or the value defined in the environmental variable COLUMNS. This default value will be overridden if the display width option (-w) is used. If the document does not define WORDWRAP, the responsibility for the document formatting is with the writer of the document.

If a file makes a link to an external file, GuideCat will do no attempt to resolve that link.

History

1.11

961119 19820 Forgot to implement options -a and -A. Now they are

there. Why the program grew this much I have no idea.

1.1

940505 19472 Replaced toupper() with own table upcase[] from file

LeoAnsi.h Now 8-bit chars should works just fine.

940428 19156 Added reading of the environmental variable $COLUMNS.

Enhanced color handling. Minor bug fixes.

940427 18116 Rewrote command parsing. Added some new command line

options, color handling, display width control and

better plain display support.

1.0

940422 15800 First working version. No external link resolving.

1.20 AmigaGuide - All Leopold-Soft knows about it

AmigaGuide - All Leopold-Soft knows about it

This is not a comprehensive and official description of the AmigaGuide format. This is just what I have figured out with the kind help of AmigaWorld (Feb-94) and my exploration of AmigaGuide documents. All terminology (like inline commands) is invented by me. If there is an official AmigaGuide document specification floating around somewhere in the net, I'd like to hear about it.

AmigaGuide documents consist of case-insensitive command lines and nodes . A node contains 8-bit Latin-1 (= ISO 8859-1) text , inline commands which of the most important is the link command..

Every document must begin with a command line , which says @DATABASE "MyFile.guide". This line will identify the file as an AmigaGuide text file.

1.21 AmigaGuide - Nodes

AmigaGuide Nodes

AmigaGuide Nodes begin with the following command line :

@NODE F_ZAPPA "The Frank Zappa discography"

F_ZAPPA is the name of the node. The text inside the quote characters will be shown as the window title for the node.

The nodes end with the command line :

@ENDNODE

Inside a node you may type standard 8-bit Latin 1 (= ISO 8859-1) text .

You may create a link to this node by adding a LINK inline command like this in any node:

The death of @{"Frank Zappa" link F_ZAPPA} really struck me hard.

As a default, a node called MAIN is considered to be the Table of Contents node, although this can be changed with the command line TOC .

1.22 AmigaGuide - Links

AmigaGuide Links

AmigaGuide links are links to other nodes in the same file or to external files.

The external files may be of any type that the Amiga OS's datatypes support: IFF ILBM images, IFF 8SVX sound samples, plain Ascii files (not AmigaGuide documents), animations and much else. They may also be links to AREXX scripts and some even more curious things.

The basic form of the link command is like this:

@{"mylink" LINK MY_NODE}

"mylink" is a text string, which AmigaGuide will present as a press button. MY_NODE is the name of the node you want the link to point to.

The external format #1 is like this:

@{"mylink" LINK MyPicture.ilbm}

This will create a link to the specified file.

If the file is a plain Ascii file, you may also use the external format #2:

@{"mylink" LINK MySource.c 25}

When pressed this link will show the file MySource.c and locate it to line #25.

If the file is an AmigaGuide document, you may use the external format #3:

@{"mylink" LINK PathFile/MY_NODE}

When pressed this will show MY_NODE from the file PathFile.

For some strange reason external links don't seem to work on XPK (2.4) compressed disk partitions.

1.23 AmigaGuide - Command Lines

AmigaGuide Command Lines

AmigaGuide command lines may be placed anywhere in an AmigaGuide document with the exception of @DATABASE, which must be the first command on the first line.

A command line must start with the "@"-character.

All commands are case-insensitive. They are capitalized in this document only to emphasize them.

All commands not recognized are treated like comments. This way an older version of AmigaGuide will not show any error messages when trying to read formatting from a newer version document.

The command lines marked with an asterisk "*" are handled by GuideCat .

The commands (I know of) are as follows:

@\$VER:MyFile.guide 1.0 (04/22/94)

An Amiga -specific version number string. This command is probably not handled as a command at all by AmigaGuide.

@(C) "Copyright ©1994 Leopold-Soft"

A copyright string. This command is probably not handled as a command at all by AmigaGuide.

@AUTHOR "Henrik Herranen"

An author string. This command is probably not handled as a command at all by AmigaGuide.

@DATABASE "MyFile.guide" *

Defines the name of the AmigaGuide file. This command must be the first command in an AmigaGuide file. "MyFile.guide" is usually the same as the name of the file.

@ENDNODE *

Ends a node .

@FONT fontname.font size

Defines the font you want to use in the next node. For some reason using this command line will fail if you type it inside a node. At least this is true with the current version of AmigaGuide (40.1).

As a rule of thumb I'd say you should not use excessively the @FONT command, because Amiga users use very different kinds of Amigas with different display modes, and they have already set their preferences to properly show their favourite font.

@INDEX INDEX_NODE

Defines which node you want to be the index node . There is a button on the AmigaGuide window called Index, which will be inactivated if this command is not used.

@NEXT NEXT_NODE

Defines which node to jump to if the user presses the "Browse >" button in AmigaGuide. This command line must be placed immediately after a @NODE (or @PREV) command line.

@NODE NODE_NAME "Window title" *

defines a beginning of a node . Nodes do not nest (what kind of an idea would that be :-)).

@PREV PREV_NODE

Defines which node to jump to if the user presses the "Browse <" button in AmigaGuide. This command line must be placed immediately after a @NODE (or @NEXT) command line.

@REM This line will demonstrate how a comment is made

A comment line. This command is probably not handled as a command at all by AmigaGuide.

@TOC TABLE_OF_CONTENTS_NODE

This will define your Table of Contents node . The default Table of Contents -node is MAIN.

@WORDWRAP *

This line tells AmigaGuide to take care of text wrapping. If you have defined this option, you may write lines as long as you wish and AmigaGuide will format them to fit on the user's AmigaGuide window. In my opinion you should always write your AmigaGuide documents with this option on. I have read that this command line may be somewhat incompatible with older versions of AmigaGuide.

1.24 AmigaGuide - Inline Commands

AmigaGuide Inline Commands

Inline commands may be inserted anywhere inside a node . They are recognized by their first two characters, "@{". The latter character makes the inline commands distinguishable from command lines when they begin a line.

All commands are case-insensitive. They are capitalized in this document only to emphasize them.

All commands not recognized are treated like comments. This way an older version of AmigaGuide will not show any error messages when trying to read formatting from a newer version document.

The command lines marked with an asterisk (*) are handled by GuideCat .

The commands (I know of) are as follows:

@{"text string" LINK NODE_NAME} *

Creates a link to a node .

@{B} *

Turns bold on. This is an example of bold text.

@{BG COLOR} *

Selects a new color for text background. COLOR is a keyword defined at the description of command @{FG COLOR}.

@{FG COLOR} *

Selects a new color for text. COLOR can be any of the following keywords:

- BACKGROUND is the default color of the backgroud (usually grey on an Amiga system and white on other systems). This is an example of BACKGROUND color text on TEXT color background, which makes this effectively reverse text.

- HIGHLIGHT selects the color the user has selected in his palette preferences as "Important text". This is an example of highlighted text. This is an example of text with a highlighted background.

- TEXT is the default text color (usually black).

@{I} *

Turns italics on. This is an example of italic text.

@{U} *

Turns underline on. This is an example of underlined text.

@{UB} *

Turns bold off.

@{UI} *

Turns italics off.

@{UU} *

Turns underline off.

1.25 AmigaGuide - Text Blocks Inside Nodes

AmigaGuide Text Blocks Inside Nodes

Nodes are filled with inline commands and text.

The text in a node is a superset of Ascii, 8-bit Latin 1 (= ISO 8859-1) (= ECMA 94) as used in Amigas, many newer Unix systems and MS Windows. It is not the same font that is used in unstandardized systems like MS-DOS, Macs or Nexts.

Because the commands begin with a "@"-character, there is an escape key provided which prevents AmigaGuide from trying to guess the meaning of the following character. This allows you to write the letter @ in your AmigaGuide documents. The escape character is "\". This will lead to the following conversions:

This: Displays this:

\@ @

\\ \

\x x

\ (nothing at all)

1.26 Words -- Counts and sorts words in a file

Words

1994-11-21

Words counts and displays how many times different words occur in a file. I don't know if this program is very useful, but I've considered it interesting enough to use it from time to time.

Installation on an Amiga System

Move the Words executable file somewhere on your default path.

Installation on a Unix System

Compile Words. With gcc it is done like this:

51% gcc -ansi Words.c -o words

52% strip words

You may also try to compile Words simply by saying make. I have included a Makefile with this package, which will also compile LeoTune and GuideCat .

Usage

Usage of Words is quite simple:

> Words [?|-h] | [-v] | [-V] | [InFile1 [InFile2 [...]]] | [-o OutFile]

If OutFile is not defined, the results are written to standard output.

If no InFiles are defined, the words are read from standard input.

The options are as follows:

-v Turn verbose mode off

-V Turn verbose mode on

-h or ? Show help

-o file Define output file (only the last definition is valid)

History

2.1

961125 13344 Recompiling code with corrected "#if defined"

directives made it slightly larger.

941121 13104 Bug fix: Words won't now choke on directories.

2.0

940506 13408 Total rewrite. Doesn't crash anymore :-) . While being

ANSI C complient the source file contains conditional

code to support Amiga pattern matching.

1.0

940505 12668 First officially published version. Just a slight

retouch from an old source file -> could be made

faster and more intelligent.

1.27 MemTest - A small and simple RAM tester

Memtest

1994-06-15

MemTest is a very simple memory checking program that scans through all free RAM memory for errors. It will only scan through memory that is considered free by the Amiga OS.

First MemTest will boost its priority to 5 to stop everything else (except interrupts) from running. Then, if started from the Workbench, it will open a console window and redirect its output to it.

The memory is checked in 64 KB chunks, or smaller, if no memory is left. All erraneous bytes will be shown. MemTest will only try to write the values $0 and $FF (0 and 255 decimal) to every location.

After the memory is checked MemTest will free its memory and return an error code to the OS. If everything went fine, MemTest returns a 0, but if it encountered any errors, 30 will be returned. However, if there are errors in your RAM, it is very likely that MemTest will crash during resource freeing, because it uses the beginnings of test areas as linked lists.

History

1.1

961125 3372 Recompiled code.

940615 3284 Switched to SAS C and replaced all <stdio.h>'s

printf()s with dos.library's Printf(). Size reduced

dramatically. Added Workbench compatibility.

Sped up the program by a factor of 4: replaced byte

tests with long word tests. The binary file got only

116 bytes longer.

1.0

940310 6308 Initial release, compiled with DICE.

1.28 WC - Word Counter

WC

1994-11-21

This program counts the lines, words and characters in each of the files specified. It's much like Unix' wc.

Usage

> Wc [?|-h] | [-l] | [-w] | [-c] | [-t] | files...

The options are as follows:

-h or ? Show help

-l Show only lines

-w Show only words

-c Show only characters

-t Show only grand total

History

1.0

961125 11760 Recompiled code.

941121 11516 First working version. Due to printing to stderr I

can't use OS 2's Printf.

1.29 Beep - The program that sez BEEP

Beep

1995-03-14

Every now and then it would be nice to let the machine do something on it's own and just let you know when it's finished. Well, why not type "Beep" in the typeahead buffer and let this marvellous program wake you up with its DisplayBeep(NULL). You know, even R. J. Mical used a program named Beep while developing the original Amiga 1000 (this was shortly before they became the Dancing Fools).

Usage

> Beep

History

1.0

950314 144 A perfect first release: No bugs, full functionality.

Right.

1.30 ScrDump - Saves screens or windows to a file

ScrDump

1996-11-25

There are many screen to disk dumper programs, but none of them that I know of can do all this:

1) Saves screens.

2) Optionally saves windows with or without borders.

3) Works with all Amigas, and all graphics cards (if used under OS 3).

4) Has compatibility fallback code for OS 2.

Usage from your Shell

> ScrDump [-w] [-s] [-b] [-B] [-h|?] [filename]

-w Save active window (only supported from OS3)

-s Save uppermost screen (default)

-b Don't save window borders

-B Save window borders (default)

-h|? Show help

filnam Specify file name to save to (default: RAM:ScrDumpFile)

After invoking ScrDump you have 10 seconds to make the desired screen highest or window active. When the image has been saved ScrDump will blink the screen you saved or where the active window was. Then it is safe to rearrange and close windows. See also Bugs!

Usage from Workbench

Just doubleclick the ScrDump icon, make the screen you want to save highest. After 10 seconds ScrDump will save the image to the file RAM:ScrDumpFile. When finished, ScrDump will blink the saved screen. See also Bugs.

About saving windows

When you are saving a window that is not the foremost one on its screen, you may sometimes get blank areas where the window wasn't visible. This is due to the way Amiga stores windows: the window may either be updated by Amiga OS or by the application responsible for the window.

If the window is maintained only by an application you will get this odd behaviour. Examples of such programs are the standard Amiga Shell and Workbench. You won't have this problem if you raise the window to the front or if the application lets Amiga OS update the window's contents.

Bugs

It is possible to close a window or a screen while ScrDump is saving it. The effects of such a prank are undefined and may be disastrous. Don't try it!

Due to the way Amiga OS manages windows memory, saving a window that is below another window doesn't always lead to the desired results. See also About Saving Windows.

History

1.1

961125 7544 Prohibited saving a window in OS2 because implementing

it properly (withour excessive trouble) is only

possible beginning from OS3.

961119 7452 A nasty bug prevented the program from working under

OS 2.x. This is now fixed. Removing code and changing

compiling options made this much smaller, too.

1.01

950420 8496 Date of saving included in the ANNO chunk.

1.0

950404 6716 The first working version. Should work with all kinds

of Amigas.

1.31 Subtitle - TV subtitler

Subtitle

1996-11-16

Subtitle is a simple TV program subtitler. If you have a genlock and wish to subtitle your favourite Babylon 5 shows to your favourite language, this is your program.

Usage

Subtitle [-f] FileName [FontName [FontSize]]

The -f option forbids Subtitle from using Finnish hyphenation.

The default FontName is CGTriumvirate and default FontSize is 39.

There is a simple file called Subtitle.example, where you can see all the formatting capabilities of Subtitle. Try this with Subtitle by issuing the command: Subtitle Subtitle.example.

After the program has started, you may enter the program itself by pressing cursor right. Now you have an empty screen. Now, you can use the following commands:

Cursor right - Next text screen (+1).

Cursor up - Repeat last text screen (± 0).

Cursor left - Previous text screen (-1).

Space - Clear current text screen.

Esc - Exit program.

Internal logics

When trying to display a text screen as well as possible, Subtitle uses the following logics:

1. It tries to fit everything in two lines.

1.1 First it tries to chop lines only at sentence boundaries.

1.2 If 1.1 failed to fit to two lines, it tries to chop lines at word boundaries.

1.3 If 1.2 failed to fit to two lines, and -f is not set, it tries to chop lines at cyllable boundaries.

2. If the text did not fit to two lines, Subtitle looks which one, 1.1, 1.2, or 1.3 gives the lowest number of total lines, and chooses the lowest numbered of them which still gives the minimum amount of lines. I.e. if 1.1 gives 4 lines and both 1.2 and 1.3 gives 3 lines, Subtitle chooses 1.2.

Some special rules where text screens can never be chopped:

1. A space that is followed by a string "-\n", where "\n" stands for the end-of-line character.

2. A space that is preceded by a string " -".

Because of this, the following spaces can never be chopped to different lines:

Kosh said so. - Boy, was that cryptic, but -

^ ^

here here

History

1.0

961116 12940 Added the "-\n" and " -" rules and option "-f".

Added NTSC support.

0.2

961114 12544 If possible, tries to put everything in two lines so

that only whole sentences are divided on different

lines. If not possible, tries to put text on two lines

cut from word boundaries. If even that fails, tries

hyphenation. If the text still can't be fit to two

lines, put it with the best-looking cutting method that

can achieve a minimum number of lines (i.e. if

sentence boundary cutting gives 5 lines and both word

and cyllable cutting gives 3 lines, use word cutting).

0.1

961027 11916 First working version. Cuts words at word and cyllable

boundaries.

1.32 LeoLib - Generic C functions

LeoLib

1994-04-19

I have included this library in this package because several LeoUtils programs use them. If you think you can use them, feel free to!

Usage How to use the library

ctype An 8-bit ctype.h replacement

History

1.33 LeoLib - Usage

LeoLib - Usage

If you want to use LeoLib in your progrmas, you must #include "LeoLib.h" in your source file.

Then, on an Amiga system, you must compile LeoLib. I have included an SMakeFile for SAS C 6.51. After you have compiled LeoLib.lib, just link LeoLib.lib with all your object files that make use of it.

On a Unix system you must first compile LeoLib. This can be easily done by simply saying make leolib.a. I have included a standard Unix Makefile with this package.

After that, you must link leotune.a with all of your programs that use LeoLib.

1.34 LeoLib - A complete 8-bit clean <ctype.h> replacement

A complete 8-bit clean <ctype.h> replacement

These functions do everything their original ANSI counterparts do, with the exception that they can fully handle the ISO 8859-1 (= ISO LATIN 1) standard character set as used in the Amiga, newer Unix systems, MS Windows etc...

The input range is from -1 to 255. For -1 (EOF) all Is-functions return 0 and To-functions -1.

int IsAlnum(c)

Returns non-zero if c is an alphanumeric character.

int IsAlpha(c)

Returns non-zero if c is an alpha character.

int IsCntrl(c)

Returns non-zero if c is a control character.

int IsDigit(c)

Returns non-zero if c is a digit.

int IsGraph(c)

Returns non-zero if c is a graphical character.

int IsLower(c)

Returns non-zero if c is a lower case letter.

int IsPrint(c)

Returns non-zero if c is a printable character.

int IsPunct(c)

Returns non-zero if c is a punctuation character.

int IsSpace(c)

Returns non-zero if c is a space character (tab, whitespace, page break, newline).

int IsUpper(c)

Returns non-zero if c is an upper case letter.

int IsXDigit(c)

Returns non-zero if c is a hexadecimal digit.

ToLower

If a character is an uppercase character, convert it to the lower case equivalent, otherwise do nothing.

ToUpper

If a character is a lowercase character, convert it to the upper case equivalent, otherwise do nothing.

1.35 LeoLib - History

History

1.0

940419 464 Initial release with an 8-bit <ctype.h> replacement.

1.36 Leopold.font

Leopold.font and LeoProp.font

1990-95

This is the best-looking font I know of for display modes with square pixels. I use it (size 9) all the time with my 20" color monitor. At least take a look at it. There are also proportional versions of the bigger sizes called LeoProp.

All fonts are copyrighted by me, but the base for two of the fonts were lent from other sources.

Credits

1993 Leopold 6 (uppercase only, chars like ¼ don't look nice)

1992 Leopold 8

1990 Leopold 9 [my system default text] (7-bit base drawn by Juha Tuominen)

1994 Leopold 11

1991 Leopold 14 (original concept from Sun workstations)

1993 LeoProp 9

1994 LeoProp 11 [my Workbench icon font]

1994 LeoProp 14 [my Workbench screen text font]

History

1.1

950404 ASCII-codes 0-31 added to non-proportional fonts.

1.0

1990-4 Fonts created.