Light, Sound & Print

Quick Reference

Disclaimer

None of us claim that these utility programs are good for anything. If **you** think they are, great, but that is up to you to decide. If any, or all, of these programs don't work, that is your problem, not ours. If you lose a million dollars, or anything else, because one or all of these programs stuffs up, you are out of pocket the million, not us. If you don't like this disclaimer: tough. We reserve the right to do the absolute minimum provided by law, up to and including nothing.

In no event will Applix be liable for direct, indirect, special, incidental, or consequential damages resulting from any defect in the software or its documentation.

This disclaimer has been provided in plain English, in keeping with Applix's policy of providing comprehensive, readable information about its products. It is basically the same disclaimer all the fancy, expensive overseas software packets provide, but without legal beagles mangling the English. Special thanks to Dave Horsfall for bringing this disclaimer to my attention.

Conditions of Sale

By purchasing† this software you agree to:

Assign Power of Attorney to the Company.

Be cited as a reference for sales purposes.

Franchise your computer to the Company.

Use only Applix brand software. Your 1616/OS EPROMS have been adjusted to operate only with Applix proprietary software, and any attempt to use any other brand will destroy your entire database, your disk drive, and the room the computer occupies.

Accept the built-in update policy. The ***modem*** ¢ program dials up Applix and checks for any upgrades. These are automatically *purchased and your credit card debited. The program earns a small sales commision.*

Deny the program is faulty. Should you find a software bug, you will be considered a dissatisfied user, who is likely to destabilise the corporate image. Should you attempt to disclose your dissatisfaction with any product or service, the Company will make every effort to preserve its credibility by destroying yours. This includes, but is not limited to, sabotaging your database, your bank accounts, your share portfolio, your credit rating, and your police records. In the unlikely event that an effective Data Protection Act ever exists, these sanctions may be replaced by hiring a hit man.

Relinquish all common law rights regarding consumer protection. Your citizenship. Your right to any intellectual property created using these programs. Your first born male child.

The site licence authorises one user at one location only to use the software. If you can afford to have two computers, you can afford to buy another copy.

Any software fault is deemed to be the fault of the user. The Help command activates a modem request for the audit of your tax returns since 1978.

The company admits no liability for the quality of its product, or for its ability to perform any function whatsoever (except act as a frisbie.)

†Aren't you glad you didn't actually **have** to purchase this shareware?

Notes on the shareware were rewritten to correspond somewhat with the actual source code, complete with gratuitous errors to confuse you, by Eric Lindsay.

Comments about this manual or the software it describes should be sent to either:

Eric Lindsay
6 Hillcrest Avenue
Faulconbridge 2776
NSW Australia
(047) 512258

Applix Pty Limited
Lot 1, Kent Street
Yerrinbool, 2575
N.S.W. Australia
(048) 839 372

ISBN 0 947341 ?? ?

A Note to Readers

This is an attempt to provide quick reference guides to the wide variety of Shareware and Public Domain programs made available for the Applix 1616 in the first seventeen Shareware disks.

It would obviously not be possible without the enthusiastic support of all of you who wrote or converted programs. In writing this material, I've generally based my descriptions on the source code, and on what has often been a very rushed play with the programs themselves. This means that I'm likely to have a lot of stuff wrong, so I'm asking for your help again.

Would you each look through my description of the programs you provided, and let me know what I have wrong, and what corrections should be made?

If any of you would like your address shown, please let me know. If you are asking for a contribution for your software, could you please let me know the suggested amount, and to where it should be sent (the usual choices are direct to you, or to you via Applix).

Some of you have provided extensive documentation files for your programs. If you would like these included in the manual, would you let me know so that I can convert and laser print them (many thanks here to Michael Johnson, Conal Walsh and Greyham Stoney, who have provided extensive documentation of their many excellent programs).

The Quibbles heading is perhaps somewhat inconsistent, as I've included suggestions for upgrades and changes. In many cases, things mentioned as bugs should be considered opportunities for others to expand the work you have done, not as things that don't work. If you think I'm in error in your case, please don't hesitate to contact me so I can make changes.

Finally, it is my firm policy on this set of manuals, to provide complete updated versions (or pages if appropriate) to **everyone** who contributes shareware for the Applix 1616.

The Shareware manuals come in three varieties.

- Games
- Light, Sound and Printers
- Utilities

This division isn't totally fixed, and is done merely to make the production of the manuals (and location of the programs) somewhat easier for me. You will notice there is no page numbering, and that programs are never listed back to back on a sheet of paper. This is intended to make it easier for you to file the pages however it suits you.

Eric Lindsay, 6 Hillcrest Avenue, Faulconbridge NSW 2776
BH (02) 2189651 Mon to Thurs. Weekends (047) 512258

60hzcrtc - 6545 video changed to 60 Hz rate - Andrew Morton - SW#6

60hzcrtc

Description

This program fiddles the CRTC 6845 video controller chip, providing 60 Hz vertical scan. It is intended to drive an Amust RGB Enhanced monitor. Should let people see how to control the chip from assembler.

Quibbles

Andrew mentions that you shouldn’t fiddle with registers 14 and 15, because he uses them on power up.

Associated files

```
60hzcrtc.exec, 60hzcrtc.s
```

See also

There is an MRD example floating round somewhere.

Distribution

Applix 1616 Shareware Disk #6 /tutorial

Author

Andrew Morton

640circle - demo of circles in 640 mode Basic - Andrew Morton - SW#4

640circle

Description

Looks like Andrew Morton decided circles in 640 mode SSBASIC were not suitable.

This is an executable routine to draw round circles in 640 resolution. It interfaces to SSBASIC. You call it as in the example code in `cir-test.bas`. It uses Brezenham's algorithm, with circles made round by dividing the *y* dimension by 2.

Examples

Call using:

```
nul% = call($4000, xcentre, ycentre, radius, colour)
```

Quibbles

Hmm, did the original SSBASIC not do it just to stay compatible with other broken BASICs?
Dave Wilson points out that the result isn't symetrical (compare straight lines at top and bottom).

Associated files

`640circle.exec, 640circle.s, circle.c, circle.doc, cir-test.bas`

See also

Distribution

Applix 1616 Shareware Disk # 4/circle

Author

Andrew Morton

1616typer - easy letter writer - Paul Cahill - SW#4

1616typer.bas

Description

An easy typewriter conversion, for doing letters and short material. Produces justified text. Enter the text using the BASIC routines. Calls the inbuilt `edit` routine if you want to make extensive changes. Ingenious.

F1	Return to main menu.
F2	Tabs to right hand side for return address.
c	Clear existing text.
e	Edit, places you in 1616 `edit`.
f	Finish, exit without saving.
l	Load a file.
p	Print a file, asks for viewing or printer.
s	Save a file to disk.
t	Text entry, with word wrap.
x	Options, prompts for single or double spacing, starting page number, lines per page, length of page. The defaults are all sensible, and easily changed in the program.

Quibbles

Extensive editing can stuff up the justification (not surprisingly).

Associated files

`1616typer.bas, 1616typer.doc`

See also

justify, **pr**, **roff**, **runoff**, **eroff**.

Distribution

Applix 1616 Shareware Disk #4 /utilities

Author

Paul Cahill

adc - sample sound via adc port - Gerhard Baumann -

adc11 `tablesize soundfile`
adc15 `tablesize soundfile`
adc22 `tablesize soundfile`
adc_s `tablesize soundfile`

Description

Sample sounds by connecting a microphone to your analog to digital converter port, input pin 27, ground pin 25. There is 11 KHz version, `adc11`, that can sample 35 seconds of sound, a 14.9 KHz version, `adc15`, that can sample about 25 seconds, a 22.3 kHz version, `adc22`, which can sample about 17 seconds. Sample limit is the size of the memory buffer you can allocate, which normally can not exceed 380 kilobyte.

The stereo version, `adc_s` samples from analogue input pins 27 **and** 28, with ground being pin 25, producing about 25 seconds at a sample rate of 7.4 KHz per channel.

Analogue inputs are sampled using a successive approximation method, with interrupts disabled. The program selects the input channel, then writes to the onboard DAC latch with only bit 7 set. The program pauses about 16 clock cycles to allow the DAC, comparator and input latch to settle, before testing the output of the comparator, which compares the output of the DAC with the voltage being sampled.

If the output is high, bit 7 is cleared, bit 6 only is set high, and the cycle repeated, until all 8 bits have been done.

The stereo version simply switches to another channel after a full sample, and does the same test there.

The external hardware consists of wires for analogue inputs 0, 1 and gound, which are connected directly to a sound source such as the output jack of a tape recorder or similar.

Remember that the maximum allowed input voltage is 2.2 volts; you can damage your input circuitry if you allow voltages in excess of this.

Quibbles

No low pass filter, therefore you may sample higher frequencies than the Nyquist limit.

Associated files

`adc.s`, `adc22.s`, etc.
sound files

See also

pic. play, wave

Distribution

Not distributed yet.

Author

Gerhard Baumann,

adc - sample sound via adc port - Michael J Sloman - SW#17

adc11 `soundfile`

Description

Sample sounds by connecting Michael’s analog to digital converter to your 1616. Details of the circuit are in µ*Peripheral*, issue 11.

Quibbles

Associated files

`adc.s`, `adc22.s`, etc.
sound files `ACDC, dreadlock, radio`
See µPeripheral, issue 11 for details of circuit.

See also

pic. play, wave

Distribution

Applix 1616 Shareware Disk #17 /bin /sFX

Author

Michael J Sloman, 1642 Main Nth Rd, Brahma Lodge SA 5109.

back - plays Mac digitised sound files backwards - Jeremy Fitz-hardinge - SW#1, SW#3

back `soundfile`

Description

Plays 11 KHz digitised sound files obtained from Macintosh. Revised assembly version of `sound11`. Uses ***getmem*** to ensure there is sufficient room in memory. Plays the sound backwards, so you can check for demonic spells in rock music.

Examples

```
back bogie.snd.11
```

Quibbles

Associated files

```
back.xrel, back.s, *.snd.11
```

See also

sound11, **sound-11**, **sound22**, **sound-22**, **adc**

Distribution

Applix 1616 Shareware Disk #3 /moresound

Author

Jeremy Fitzhardinge

back11 - plays Mac digitised sound files backwards - Jeremy Fitz-hardinge - SW#1, SW#6

back11 `soundfile`

Description

Plays 11 KHz digitised sound files obtained from Macintosh. Revised assembly version of `sound11, back`. Uses ***getmem*** to ensure there is sufficient room in memory. Checks that you have Version 3 EPROMS. Highly similar to `back`.

Plays the sound backwards, so you can check for demonic spells in rock music.

Examples

```
back bogie.snd.11
```

Quibbles

Associated files

`back11.xrel, back11.s, *.snd.11`

See also

sound11, **sound-11**, **sound22**, **sound-22**, **back**

Distribution

Applix 1616 Shareware Disk #6 /soundstuff

Author

Jeremy Fitzhardinge

back22 - plays Mac digitised sound files backwards - Jeremy Fitzhardinge - SW#1, SW#6

back22 `soundfile`

Description

Plays 22 KHz digitised sound files obtained from Macintosh. Revised assembly version of `sound11, back`. Uses ***getmem*** to ensure there is sufficient room in memory. Checks that you have Version 3 EPROMS. Highly similar to `back`.

Plays the sound backwards, so you can check for demonic spells in rock music. Gives better quality than 11 KHz, at the expense of occupying twice the memory for a given sound sample.

Examples

```
back bogie.snd.11
```

Quibbles

Jeremy didn’t change his comments when he changed the rate of this, so the source is virtually identical to `back11.s`.

Associated files

`back22.xrel, back22.s, *.snd.22`

See also

sound11, **sound-11**, **sound22**, **sound-22**, **back**

Distribution

Applix 1616 Shareware Disk #6 /soundstuff

Author

Jeremy Fitzhardinge

clrwin

Description

Tutorial on creating and clearing an 80 x 25 window using system calls from assembler.

Quibbles

Associated files

`clrwin.exec, clrwin.s`

See also

diskio, **prog?**

Distribution

Applix 1616 Shareware Disk #6 /tutorial

Author

Sid Young

colour-bars

Description

Draws pretty pictures of 16 coloured vertical bars on the 1616 screen. Used for determining correct settings of a grey scale monochrome monitor, and to select (readable) appropriate colours for programs.

Quibbles

Associated files

`colour-bars.s, colour-bars.doc`

See also

kal

Distribution

Applix 1616 Shareware Disk #5 /colour-bars

Author

Michael Johnson

convert - Amiga sound files to 1616 - Matthew Geier - SW#15

convert `filein fileout`

Description

Converts sound files from the Commodore Amiga to Applix 1616 sound formats, and vice versa.

The Apple Macintosh and the Applix 1616 sound files are 8 bit unsigned integers, and have 128 as the 0 volt level, with the most negative voltage at 0, and the most positive voltage at 255. The Amiga has *signed* 8 bit data, so -127 is the most negative voltage, +128 is the most positive. All the program has to do is invert bit 7.

Quibbles

Associated files

`convert.c, convert.xrel`

See also

sound, play, etc.

Distribution

Applix 1616 Shareware Disk #15 /geier

Author

Matthew Geier, 24 June 1989.

crtsav - MRD blanks the display - Jeremy Fitzhardinge - SW#3

crtsav `[time]`

Description

This program must be included in your MRDRIVERS file. It blanks the display after an optional number of seconds (remember that numbers are in hexadecimal, if no decimal point is used). While the display is blanked, it toggles the motherboard LED every two seconds.

If invoked with no arguments, it gives the current delay setting. If invoked with **0**, it is disabled.

Pressing both shift keys together will also blank the screen, while this MRD is installed. Pressing any other key restores the screen.

Associated files

`crtsav.mrd, crtsav.s`

Distribution

Applix 1616 Shareware Disk #3 /utilities

Author

Jeremy Fitzhardinge

dissolve - clear the display randomly - Cameron Hutchison - SW#6

dissolve

Description

Clears the display by randomly setting all pixels to black, over a period of a few seconds. Nice for special effects.

Cameron has done several different versions, however those using the ***syscalls*** were too slow, so version 1.2a accesses the video ram direct. Good comments in the source code.

Quibbles

Associated files

`dissolve.xrel, dissolve.s`

See also

termite

Distribution

Applix 1616 Shareware Disk #6 /utilities

Author

Cameron Hutchison
Random number algorithm from Nov 1986 *Dr Dobbs*.

doc-write - menu system for dde editor - Michael Johnson - SW#16

doc-write

Description

`Doc-write` is a menu driven front end for Andrew Morton’s `dde` text editor. It is derived from Michael Johnson’s `easy_write`, which is a menu driven front end for editing and compiling C programs. The program requires access to a data file, `doc-write.data`, which stores various parameters for editing, and also stores a list of the filenames associated with this invocation of the editor. This allows you to group together a set of related documents, and keep track of them as a group.

The menu has provision for editing a file, printing a file to either display or printer, adding, deleting or changing a filename, and setting various modes for processing.

The edit menu lists all files currently used by `doc-write` in the current application, so you can select the one you want. The print selections will prompt for the start and end page numbers (default is print all pages). Delete and change filenames will display all current files, for your selection.

The set mode menu allows you to change the `dde` editor parameters, via a menu for each such parameter. These parameters include autoindent mode, document or non-document mode, insert or overwrite, justification, left and right margins, and tab spacing.

The program stores parameter information in a file called `doc-write.data`.

Quibbles

Changing a name on the menu doesn’t change the name of the associated file ... pity. Needs access to a search routine like `grep`, for when you can’t recall which file contains the thing you want to alter.

Associated files

`ccdoc-write.shell, doc-two.c, doc-two.obj, doc-write.c, doc-write.data, doc-write.doc, doc-write.err, doc-write.h, doc-write.obj, doc-write.xrel, easy_write.data`

See also

easy_write

Distribution

Applix 1616 Shareware Disk #16 /Johnson

Author

Michael Johnson, 11/12 Kokoda Street, Wagga Wagga, NSW 2650, (069) 255255. This is Version 1.0, dated 19 July 1989.

dumpbro - print screen to Brother M1109 dot matrix - Michael Sloman - SW#6

dumpbro

Description

Dumps the contents of the 640 mode video display, to a Brother M1109 dot matrix printer running in IBM mode on the parallel printer port. The code is short and clean, but the ***syscall*** used may slow it down.

Quibbles

Assumes you use highest 32k page for video (usually true). It would be nice to have optional serial output (the M1109 does have a serial port), perhaps as a command line switch. Needs to be running as an MRD, triggered by the PrtSc key.

Associated files

`dumpbro.exec, dumpbro.s`

See also

dumpscreen

Distribution

Applix 1616 Shareware Disk #6 /utilities

Author

Michael Sloman, 1642 Main Nth Road, Brahma Lodge SA 5109

dumpch - display character shapes in hex - Sid Young - SW#6

dumpch

Description

Reads in the 1616 character set, and prints the bit pattern that draws each in hexadecimal. Redirect to file for editing and modification.

Quibbles

I think this is incomplete. Looks like Sid probably intends to add a modify routine later.

Associated files

`dumpch.exec, dumpch.s`

See also

invertch

Distribution

Applix 1616 Shareware Disk # 6/utilities/sid

Author

Sid Young

dumpscreen - graphics printer screen dump - Norm Clark - SW#3

dumpscreen

Description

Dumps the contents of the screen to a Star SG15 dot matrix printer. Includes notes for converting the output for an Epson FX80+ printer.

Assumes a 640 by 200 display, not 320 by 200. The result is 640/72 inches by 200 x 2/72 inches.

Quibbles

I have a nasty suspicion that this needs to be redone as an MRD, so you can call it by using the PrtSc key from within other programs.
A 320 x 200 version would be nice.
A colour version would be even nicer.
Versions for other brands of printer would be welcome.

Associated files

`dumpscreen.exec, dumpscreen.s`

See also

format, **justify**, **dumpbro**

Distribution

Applix 1616 Shareware Disk #3 /printer

Author

Norm Clark

emacs - powerful minicomputer editor - Colin McCormack - SW#14

emacs

Description

Mainframe editor.

Make a directory of `emacs` related files.

`Assign /emacs` to this directory.

Put the file `ansifk.mrd` into your `mrdrivers` file. This enables the function keys and cursor pad - `ansifk` emulates IBM function keys.

Arrange that the command `ansifk +` be ececuted before `emacs`.

Load the extensive tutorial, and play around.

You are looking at the MicroEMACS tutorial.

NOTE: This tutorial attempts to help you "learn by doing". The characters ">>" at the left margin of your screen indicate directions for you to try using a command.

EMACS commands generally involve the CONTROL key (sometimes labelled CTRL or CTL) or the META key (generally labelled ESCAPE). Rather than write out CONTROL or META each time we want you to prefix a character, we'll use the following abbreviations:

^<chr>   Hold the CONTROL key while pressing the character <chr>. Thus, ^F would be: hold the CONTROL key and press F.

>> Now type ^V (View Next Screen) to move to the next screen. Remember: hold the CONTROL key and press V.

ESC-<chr> Press the ESCAPE key and release it, then press the character <chr>. Note: The command will have the same meaning for upper or lower case characters (<chr>).

IMPORTANT NOTE: If you must exit at some point, type ^X^C.

For the time being, you'll be expected to type ^V whenever you finish reading the current screen.

Note that there is an overlap when going from screen to screen; this provides some continuity when moving through the file.

The first thing that you need to know is how to move around from place to place in the file. You already know how to move forward a screen with ^V. To move back a screen, type ^Z.

>> Try typing ^Z and then ^V to move back and forth between screens a few times.

SUMMARY

The following commands are useful for viewing screens:

^V	Move forward one screen
^Z	Move back one screen
ESC-^L	Clear screen and redisplay everything, putting the text near the cursor at the center of the screen.

>> Find the cursor and remember what text is near it. Type an ESC-^L. Find the cursor again and see what text is near it now.

BASIC CURSOR CONTROL

Getting from screen to screen is useful, but how do you reposition yourself within a given screen to a specific place? There are several ways you can do this. One way (not the best, but the most basic) is to use the commands previous, backward, forward and next. As you can imagine these commands (which are given to EMACS as ^P, ^B, ^F, and ^N respectively) move the cursor from where it currently is to a new place in the given direction. Here, in a more graphical form, are the commands:

```
                  Previous line, ^P
                        :
                        :
  Backward, ^B .... Current cursor position .... Forward, ^F
                        :
                        :
                  Next line, ^N
```

You'll probably find it easy to think of these by letter. P for previous, N for next, B for backward and F for forward. These are the basic cursor positioning commands and you'll be using them ALL the time so it would be of great benefit if you learn them now.

>> Do a few ^N's to bring the cursor down to this line.

>> Move into the line with ^F's and then up with several ^P's. Note what ^P does when the cursor is in the middle of the line.

>> Try ^B at the beginning of a line. Note what happened to the cursor. Do a few more ^B's. Then do ^F's back to the end of the line and beyond.

When you go off the top or bottom of the screen, the text beyond the edge is shifted onto the screen so that your instructions can be carried out while keeping the cursor on the screen.

>> Move the cursor off the bottom of the screen with ^N's and see what happens. Note the new position of the cursor.

If moving by characters is too slow, you can move by words. ESC-F moves forward a word and ESC-B moves back a word.

>> Type a few ESC-F's and ESC-B's. Intersperse them with ^F's and ^B's.

Notice the parallel between ^F and ^B on the one hand, and ESC-F and ESC-B on the other hand. Very often META characters are used for operations related to English text whereas CONTROL characters operate on the basic textual units that are independent of what you are editing (characters, lines, etc.).

Two other commands which are useful are ^A and ^E. These commands move the cursor to the beginning (^A) and the end (^E) of the line.

>> Try a couple of ^A's, and then a couple of ^E's. Note that the cursor does not move when either of these commands is repeated continuously.

Two other simple cursor motion commands are ESC-< (less than), which moves to the beginning of the file, and ESC-> (greater than), which moves to the end of the file. If you need the shift key to type a "<", then you must also use the shift key to type ESC-<. Otherwise, you would be typing ESC-, .

The location of the cursor within the text is also called "point". To paraphrase, the cursor shows on the screen where point is located in the text.

Here is a summary of simple moving operations, including the word and line moving commands:

^F	Move forward a character
^B	Move back a character
ESC-F	Move forward a word
ESC-B	Move back a word
^N	Move to next line
^P	Move to previous line
ESC-N	Move to next paragraph
ESC-P	Move to previous paragraph
^A	Move to beginning of line
^E	Move to end of line
ESC-<	Go to beginning of file
ESC->	Go to end of file

>> Try all of these commands now a few times for practice as these are the most often used commands. Since the last two will take you away from this screen, use ^V's and ^Z's to return here.

Like all other commands in EMACS, these commands can be given arguments which cause them to be executed repeatedly. The way you give a command a repeat count is by pressing META (ESC) and then the number before you enter the command. As a special case, typing ^U is equivalent to ESC-4.

For instance, ESC-8 ^F moves forward eight characters.

>> Try giving a suitable argument to ^N or ^P to come as close as you can to this line in one jump.

This also applies to the screen moving commands, ^V and ^Z. When given an argument, they scroll the screen up or down by that many screens.

>> Try typing ESC-3 ^V now.

If you would like to scroll up, you can give an argument to ^Z.

ABORTING COMMANDS

The EMACS command used to abort any command which requests input is ^G. For example, you can use ^G to discard a numeric argument or at the beginning of a command that you don't want to finish.

>> Type ESC-100 to make a numeric argument of 100, then type ^G. Now type ^F. How many characters does it move? If you have typed an ESC by mistake, you can get rid of it with ^G^G.

ERRORS

Sometimes you may do something which EMACS doesn't allow. If it is something simple, such as typing a CONTROL key sequence which is not associated with any command, EMACS will just beep at you. Otherwise, EMACS will also display an informative error message at the bottom of the screen.

Some versions of EMACS do not have all the features described in this tutorial implemented yet. If you come across such an unimplemented feature, you may get an error message when you try to use it. Just press any cursor movement key and proceed to the next section of the tutorial.

NOTE: Several of the exercises in the following sections allow you to use options which will make changes to this tutorial. Do no worry about these changes affecting the tutorial - this is only a copy of the master tutorial and you will not be instructed to save the changes made to it.

CURSOR KEYS

The cursor keypad, usually located on the right side of the keyboard, has been bound to some of the more useful screen movement commands. The mappings are as follows:

Cursor-Right	^F	Move forward a character
Cursor-Left	^B	Move back a character
^Cursor-Right	ESC-F	Move forward a word
^Cursor-Left	ESC-B	Move back a word
Cursor-Down	^N	Move to next line
Cursor-Up	^P	Move to previous line
Pg-Dn	^V	Move to next screen
Pg-Up	^Z	Move to previous screen
Home	ESC-<	Go to beginning of file
End	ESC->	Go to end of file
Insert	^C	Insert single space
Delete	^D	Delete current character

A map of the keypad layout looks something like this:

```
            --------------------------------------------------
            | 7          | 8          | 9          |
            | Home       | ^          | Pg Up      |
            |     ESC-<  | |      ^P  |        ^Z  |
            --------------------------------------------------
            | 4          | 5          | 6          |
            | <--     ^B |            | -->     ^F |
            --------------------------------------------------
            | 1          | 2          | 3          |
            | End        | |          | Pg Dn      |
            |     ESC->  | v      ^N  |        ^V  |
    -------------------------------------------------------------
    | 0                      | .                      |
    | Insert              ^C | Delete              ^D |
    -------------------------------------------------------------
```

>> Practice using the cursor keypad.

MODE LINE

The line above the function key display at the bottom of the screen is referred to as the "communication line". This is where EMACS interactively communicates with you. Later you will see how EMACS prompts you for information on this line, such as to initiate a search. EMACS can report things to you on this line as well.

>> Type ^X= and see what appears in the communication line. Don't worry about what all this information means - it is just an example of how EMACS lets you know more about the file you are editing.

The line immediately above the communication line is referred to as the "mode line". The mode line looks something like

```
=*== MicroEMACS 3.9i () == emacs.tut == emacs.tut
```

This is a very useful "information" line.

- The asterisk (star) indicates that changes have been made to the file. Immediately after opening or saving a file, there is no star.

- Any words inside the parentheses indicate the "modes" EMACS is currently in. Modes will be discussed in the next section.

- The string following the () is the buffername, i.e., the name EMACS gives to the buffer, and it is usually related to the filename.

- The string following "File:" is the name of the file you are currently editing.

>> Look at the mode line and identify the items discussed above.

MODES

Listed within the parentheses are the "modes" which are associated with the current buffer. Modes are a feature of EMACS which assist in the editing of different languages, i.e., C, and text. Presently, there are no modes associated with this buffer. This means EMACS will do exactly what you think it will when using it - no "bonuses". You can find out more about the current buffer and mode status by typing ^X^B. Refer to the EMACS manual for a further discussion of buffers and modes.

As you become more familiar with EMACS and the use of buffers, "mode" takes on additional meaning. When more than one buffer is in use, a mode is referred to as "local" or "global". These terms indicate how a mode will affect the current buffer and other existing or to be added buffers.

A "local" mode is valid only within the scope of the current buffer. Other existing buffers and buffers which will be added are not affected by local modes.

The commands to add and delete local modes are

^XM	Add a local mode
^X^M	Delete a local mode

Each of the above commands will prompt you for a mode. To activate (deactivate) a mode, type the name of a valid (active) mode (refer to EMACS manual for a complete list of the valid modes) and follow it by pressing <Return>, the carriage-return key.

>> Type ^XM WRAP - note the change in the mode line. Move the cursor to a blank line on this screen and begin typing the sequence "asdf ". Continue typing this sequence and note what happens when the right margin is encountered.

The previous exercise allowed you to enter text with the "WRAP" mode active. As you can see, "WRAP" instructs EMACS to break between words when a line gets too long. However, in order for this mode to be effective, spaces must be inserted between words.

The right margin is usually set at 72 characters but it can be changed. To change the margin type ESC nn ^XF where "nn" is the column number of the new right-hand margin.

>> Type ESC 40 ^XF. Then begin typing "asdf " and notice where the line now breaks. To return to the default right-hand margin, type `ESC 72 ^XF`.

>> Type ^X^M WRAP to "turn off" the local mode "WRAP".

A "global" mode affects only those buffers which will be ADDED after the "add/delete global mode" command is executed - not the current or other existing buffers. Currently there is no global mode set.

The commands to add and delete global modes are

ESC-M	Add a global mode
ESC-^M	Delete a global mode

Note: All modes can be local. However, global modes allow you to activate those modes which usually apply to most of the buffers in use.

As with local modes, each of the above commands will prompt you for a mode. To activate (deactivate) a mode, enter the name of a valid (active) mode.

>> Type ESC-M OVER. This mode tells EMACS to write over the text on the current line. Is there any change in the mode line? Now move to the line of "asdf " you entered and start typing. Note that nothing happens. Remember that global modes affect only those modes which will be added - not those already existing.

>> Type ESC-^M OVER to "turn off" the global overwrite mode.

INSERTING AND DELETING

If you want to type text, just start typing. Characters which you can see, such as A, 7, *, etc. are taken by EMACS as text and are immediately inserted. Type <Return> to insert a line separator, i.e., a single linefeed character.

You can delete the last character you typed by typing either <Delete> or ^H. On some keyboards, there is a dedicated key for creating a ^H. If so, it is usually labelled as either "Backspace" or "<--". <Delete> is a key on the keyboard, which may be labelled "Rubout" instead of "Delete" on some terminals. More generally, <Delete> deletes the character immediately before the current cursor position.

>> Now type a few characters and then delete them by typing <Delete> a few times.

>> Now start typing text until you reach the right margin, then continue to type. When a line of text gets too big for one line on the screen, the line of text is "continued" off the edge of the screen. The dollar sign at the right margin indicates a line which has been continued. EMACS scrolls the line over so you can see what you are editing. The "$" at the left or right edge of the screen indicates that the current line extends off in that direction.

This concept is easier to understand by doing rather than by reading about it so it is suggested that the following exercises be done.

>> The following line actually goes off the edge. Try typing enough ESC-F's so that you move off the right hand end of this line. This is a long line of text. Note the "$" at each edge. Keep typing ESC-F's and watch where EMACS decides to scroll the line. Now, type ESC-B's until EMACS decides to scroll the line again.

>> Go to the line you entered which the text continued off the edge of the screen. Use ^D's to delete the text until the text line fits on one screen line again. The continuation "$" will go away.

>> Move the cursor to the beginning of a line and type <Delete>. This deletes the line separator before the line and merges the line onto the previous line. The resulting line may be too long to fit on the screen, in which case it has a continuation indicator.

>> Press <Return> to insert the separator again.

Internally, EMACS will allow you to have lines of nearly any length, limited only by the amount of memory available. Externally, however, EMACS can only read or write lines, to or from a file, which are less than or equal to 255 characters.

Remember that most EMACS commands can be given a repeat count. Note that this includes characters which insert themselves.

>> Try that now -- type ESC-8 * and see what happens.

If you want to insert spaces in a line, type ^C.

>> Move to a line and move the cursor with ^F's; then insert spaces with ^C. Use ^D to remove the spaces.

If you want to create a blank line between two lines, move to the second of the two lines and type ^O.

>> Try moving to a line and typing ^O now.

You've now learned the most basic way of typing something in EMACS and correcting errors. You can delete characters, words or lines as well. Here is a summary of the delete operations:

<Delete>	Delete the character just before the cursor
^H	Delete the character just before the cursor
^D	Delete the character the cursor is under
ESC-<Delete>	Kill the word immediately before the cursor
ESC-^H	Kill the word immediately before the cursor
ESC-D	Kill the word from the cursor position
^K	Kill from the cursor position to end of line

Notice that <Delete> and ^D vs ESC-<Delete> and ESC-D extend the parallel started by ^F and ESC-F (well, <Delete> isn't really a control character, but let's not worry about that).

Now suppose you kill something, and then you decide that you want to get it back? Well, whenever you kill something bigger than a character, EMACS saves it for you. To yank it back, use ^Y. Note that you don't have to be in the same place to do ^Y. This is a good way to move text around. Also note the difference between "Killing" and "Deleting" - "Killed" text can be yanked back, and "Deleted" text cannot. Generally, the commands that can destroy a lot of text save it, while the ones that attack only one character do not save it.

>> Type ^N a couple times to position the cursor at some line on this screen. Now kill that line with ^K.

Note that a single ^K kills the contents of the line, and a second ^K kills the line itself, and makes all the other lines move up. If you give ^K a repeat count, it kills that many lines AND their contents.

The text that has just disappeared is saved so that you can retrieve it. To retrieve the last killed text and put it where the cursor currently is, type ^Y.

>> Try it. Type ^Y to yank the text back.

Think of ^Y as if you were yanking something back that someone took away from you. Notice that if you do several ^K's in a row the text that is killed is all saved together so that one ^Y will yank all of the lines.

>> Try it. Type ^K several times.

>> To retrieve that killed text: Type ^Y. Move the cursor down a few lines and type ^Y again. You now know how to copy text.

What do you do if you have some text you want to yank back, and then you kill something else? ^Y would yank the more recent kill.

>> Kill a line, move around, kill another line. Then do ^Y to get back the second killed line.

SEARCHING

EMACS can do searches for strings (these are groups of contiguous characters or words) either forward through the file or backward through it.

>> Now type ^S to start a search. Type the word "cursor", then ESC.

>> Type ^S ESC to find the next occurrence of "cursor".

The ^S starts a search that looks for any occurrence of the search string AFTER the current cursor position. But what if you want to search for something earlier in the text? To do this one should type ^R for Reverse search. Everything that applies to ^S applies to ^R except that the direction of the search is reversed.

TEXT REPLACEMENT

>> Move the cursor to the blank line two lines below this one. Then type ESC-R changed ESC altered ESC .

Notice how this line has changed; you have replaced the word "changed" with "altered" wherever it occurs in the file after the cursor. After all the substitutions have been made or the end of file has been reached, a message informing you of the number of substitutions which have been made appears in the communication line.

The more customary command for replacing strings is the interactive command query-replace-search (ESC-^R), which has several options. In essence, it shows each occurrence of the first string and asks you if you want to replace it or not. Type a "?" when it asks to replace the string to list the various options for query-replace-search. For a more detailed discussion of this command refer to the EMACS manual.

FILES

In order to make the text changes permanent, you must save them to a file. If you do not save them, the changes will "disappear" when you leave EMACS. As you make changes, i.e., corrections, deletions, insertions, etc., they are actually written to a "scratch" copy of the file and the changes to this file will not affect the "master" copy of the file until a file save is specified. This allows you to decide if changes made to the file should be made permanent or discarded.

Remember: The file name appears on the mode line.

```
=*== MicroEMACS 3.9i () == emacs.tut == File: emacs.tut
```

The commands for finding and saving files are unlike the other commands you have learned so far in that they consist of two characters - a ^X followed by another character which specifies the file command to be executed.

To find a file, type ^X^F. EMACS will then prompt you from the communication line for the name of the file. In response to the prompt, type the file name followed by a <Return> to indicate the file name has been entered. This command will tell EMACS to go find this file and load it. Its contents will then be displayed on the screen and you will be able to edit the file's contents.

To save any changes made to the file, type ^X^S. This tells EMACS to create a new version of the file which includes the changes you have made. When the save is complete, the number of lines saved will be displayed in the communication line.

If you edit a file and at some point decide to quit (i.e., ^X^C) without saving the changes, EMACS will remind you that changes have been made to the file and ask you if you really want to quit. Enter "N" to return to EMACS or "Y" to exit EMACS without saving the changes.

To create a file, just edit it "as if" it already existed. Then start typing in the text. When you ask to "save" the file, EMACS will really create the file with the text that you have entered. From then on, you can consider yourself to be editing an existing file.

It is not easy for you to test editing a file and continue with the tutorial. But you can always come back into the tutorial by starting it over and skipping forward. So, when you feel ready, you should try editing a file named "FOO", putting some text in it, and saving it; then exit EMACS and look at the file to be sure that it worked.

EXTENDING THE COMMAND SET

There are many, many more EMACS commands than could possibly be put on all the CONTROL and META characters. EMACS gets around this with the X (eXtend) command. There are two forms of this command:

^X	Character eXtend. Followed by one character.
ESC-X	Named command eXtend. Followed by a long name.

These are commands that are generally useful but used less than the commands you have already learned about. You have already seen two of them: the file commands ^X^F to Find and ^X^S to Save. Another example is the command to tell EMACS that you'd like to stop editing. The command to do this is ^X^C.

There are many ^X commands. Right now, the most helpful ones will be

^X^F	Find file.
^X^S	Save file.
^X^C	Quit EMACS. This does not save your files automatically; however, if your files have been modified, EMACS asks if you really want to quit. The standard way to save and exit is ^X^S ^X^C.

Named eXtended commands are commands which are used even less frequently, or commands which are used only in certain modes. These commands are usually called "functions". An example is the function "apropos", which prompts for a keyword and then gives the names of all the functions that are apropos for that keyword. When you type ESC-X, EMACS prompts you from the communication line with ":" and you should type the name of the function you wish to call; in this case, "apropos". Just type "apr<Space>" and EMACS will complete the name. EMACS will ask you for a keyword or phrase and you type the string that you want information on.

>> Type ESC-X, followed by "apropos<Return>" or "apr<Space>". Then type "file" followed by a <Return>. Note: ESC-A is equivalent to the ECS-X "apropos" command.

>> To remove the "window" that was added, type ^X0 (zero).

FUNCTION KEYS

By now, you should be familiar with the format and meaning of some of the more common CONTROL and META commands. Because several of these commands are used frequently, they have been bound to the function keys, which are usually located on the left-hand side of the keyboard and labelled F1..F10. By pressing the appropriate function key, one can replace several keystrokes with a single keystroke, thus saving you time as you become familiar with their use.

The highlighted portion at the top of the screen lists the commands which are associated with each function key. Each function key supports two commands specified by fn or Fn where n = 1, 2,...10. The default commands are represented by fn and are defined on the left side of the screen; these commands are executed by pressing the appropriate function key. The secondary commands are represented by Fn and are defined on the right side of the screen; these commands are executed by pressing the <Shift> key and the appropriate function key at the same time.

>> Press f1 would ESC. Note the position of the cursor - "would" was located just as if ^S would ESC had been entered. Enter ^S would ESC to see for yourself.

>> Press F1 (<Shift> f1). Note the different appearance of the screen. You have toggled the function key list, i.e., "turned it off". To "turn it on", press F1 again.

>> Try using some of the other function keys to become familiar with their use. NOTE: Do NOT use f9 with this file as it would save any changes you may have made while using the tutorial.

GETTING MORE HELP

In this tutorial we have tried to supply just enough information to get you started using EMACS. There is so much available in EMACS that it would be impossible to explain it all here. However, you may want to learn more about EMACS since it has numerous desirable features that you don't know about yet.

The most basic HELP feature is the describe-key function which is available by typing ^X? and then a command character. EMACS prints one line in the communication line to tell what function is bound to that key.

>> Type ^X?^P. The message in the communication line should be something like "^P is bound to previous-line".

NOTE: Multi-character commands such as ^X^Z and ESC-V are also allowed after ^X? .

The describe-command function does not work - December 1986
Skip to the next section

The describe-command function (ESC-?) will prompt for the name of a function and print out the section from the manual about that command. When you are finished reading it, type a space or a ^G (quit) to bring your text back on the screen.

Now let's get more information about the previous-line command.

>> Type ESC-?^P. When you are finished reading the output, type <Space>.

The "name of the function" is important for people who are customizing EMACS. It is what appears in the EMACS CHART as the documentation for the command character.

CONCLUSION

Remember: To EXIT use ^X^C.

This tutorial is meant to be understandable to all new users, so if you found something unclear, don't sit and blame yourself - complain!

You'll probably find that if you use EMACS for a few days you won't be able to give it up. Initially it may give you trouble. But remember,this is the case with any editor, especially one that can do many, many things - and EMACS can do practically everything.

ACKNOWLEDGEMENTS

This is a modified version of the "JOVE Tutorial" by Jonathan Payne (19 January 86). That document was in turn a modified version of the tutorial "Teach-Emacs" from MIT as modified by Steve Zimmerman at CCA-UNIX (31 October 85).
Update - February 1986 by Dana Hoggatt.

Update - December 1986 by Kim Leburg.
Update - November 1987 by Daniel Lawrence
Update - February 1988 by Daniel Lawrence

Quibbles

Associated files

`.emacsrc, ansifk.c, ansifk.mrd, bpage.cmd, emacs.hlp, emacs.ref, emacs.tut, emacs.txt, emacs.xrel, newpage.cmd, ppage.cmd, read.bak, read.me, wpage.cmd`

See also

vi

Distribution

Applix 1616 Shareware Disk #14 /emacs 433 blocks

Author

Ported to 1616 by Colin McCormack

fft

Description

Perform forward or inverse Fast Fourier Transform, length 2**n.

`fft` is a function which forward transforms time series real data to frequency speectrum complex data, or the inverse, where the length of the data is a power of 2.

Synopsis

```
fft   (a,  b,  m,  dx,  inverse)
      double a[];
      double b[];
      int m
      double dx;
      int inverse;
```

a is the input and output array of real values, either time series or frequency for inverse transform. **b** is the input and output array of imaginary values, 0 for forward, frequency for inverse. **m** is the log (base 2) of hald the length of a and b. **dx** is the time between samples, and inverse is true only for an inverse transform.

Quibbles

No executable on SW#9.

Associated files

`fft.3, fft.c, fft.man, fftest.c, pulse, saw, square, triangle`

See also

Distribution

Applix 1616 Shareware Disk #9 /fft

Author

Dale Carstensen, Los Alamos National Laboratory
Ported to 1616 by Andrew Morton

format - printer utility, for dot matrix - Andrew Morton, Dale Barnes - SW#3

format `filename` [*n*]

Description

Prints *n* copies (where *n* is in hexadecimal, or **.***n* for decimal) of a file containing printer format commands. The format commands included in the file are translated to appropriate commands for common Epson printers (you should change the program to suit your own printer). Most of the commands are **control** characters, usually **^Z**, followed by a single letter which indicates what is to be done.

^**Ze**	emphasised print
^**Zf**	turn off emphasised print
^**Z-**	underline on
^**Z+**	turn off underlining
^**Z8**	paper out sensor off (for single sheet feeding)
^**Zg**	double strike, for bolder printing
^**Zs**	superscript on
^**Zz**	subscript on
^**Zt**	turn off subscript or superscript
^**Zh**	turn off underline, subscript or superscript
^**O**	condensed text
^**R**	turn off condensed text
^**L**	page feed

Examples

format `text 12`
prints 18 copies. If the file `text` contains

```
^Ze ^Z- emphasised underlined ^Zf ^Z+
^Zg ax^Zs2^Zt = ay^Zs2^Zt in double strike^Zh
```

then the result should be

emphasised <u>underlined</u>
ax^2 = ay^2 in double strike

Quibbles

Assumes a parallel printer only.
Probably be better to send to standard output if you also have serial printers.

Associated files

`format.exec, format.s, format.doc`

See also

justify, **dumpscreen**, **1616typer**, **roff**, **eroff**, **runoff**

Distribution

Applix 1616 Shareware Disk #3 /printer/format

Author

Original version by Andrew Morton
This version by Dale Barnes

frac - display fractal generated images - Stephen Nicholson - SW#6

frac `datafile`

Description

Draws pretty pictures, using fractals. Based on an article in Scientific American (don't ask me where ... I didn't see the article).

alt s	to stop picture generation
esc	to save picture to a disk (prompts for name)
alt c	exit the program
	continue using any other key

Examples

```
frac tree
```

Quibbles

The white dots on blue background that Stephen choose are almost invisible on my colour monitor. Look in the source code for `trap7(37,3,15)` and change it to some colour you can see.

You can't dump the picture back on the display correctly with `mload filename 78000` (hmm, wonder if it is resolution or byte shuffling?)

Associated files

`frac.exec, frac.c, frac.obj,`
`fern, square, tree, triangle` (data files).

See also

kal, **life**

Distribution

Applix 1616 Shareware Disk #6 /frac

Author

Stephen Nicholson

ft - freetone experimental sounds - Sid Young - SW#16

ft etc

Description

Short programs that make nice little sound effects. Make sure a loudspeaker is attached to your 1616, then just run the program. Written 31/10/1988.

Quibbles

I can’t think of any decent way to describe all the effects.
`ft` dot, dot, dot
`ft1` twinkle
`ft2` dit, dit, dit
`ft3` beep, beep, beep
`ft4` ditditdit, dotdotdot
`ft5` drr, drr, drr
`ft6` slowdown
(I told you I couldn’t describe them!)

Associated files

`ft.s, ft1.s, ft2.s, ft3.s, ft4.s, ft5.s, ft6.s`

See also

sound, **wave**, etc

Distribution

Applix 1616 Shareware Disk #16 /syd

Author

Sid Young

imagewriter - prints runoff format files - Craig Mills - SW#10

imagewriter `filename.roff`

Description

Takes a `.roff` text file formatted with commands for `runoff`, and prints it with bold, underline, enlarged, etc., fonts. Full details under `runoff`. An impressive and extensive piece of assembler work.

Quibbles

Substitute for `dde` editor, if you haven't bought a copy.

Associated files

`imagewriter.exec, runoff.exec, /chsets/*.chsets`

See also

runoff

Distribution

Applix 1616 Shareware Disk #10 /bin

Author

Craig Mills

invertch

Description

Modify the entire character set by inverting each pixel (black for white, white for black).

It certainly looks different! I forgot to test it with a coloured character set.

Quibbles

Seems to me it would be easier to get the same appearance using ***syscall .34***

Associated files

`invertch.exec`, `invertch.s`

See also

dumpch

Distribution

Applix 1616 Shareware Disk # 6/utilities/sid

Author

Sid Young

justify - justify a text file - Andrew Morton, Norm Clark - SW#3

justify `filename`

Description

Justifies text in a file to 80 characters per line and directs output to a parallel printer. If a line contains less than 70 characters, it is not justified.

Quibbles

Assumes a parallel printer.
Is somewhat dumb about short lines.
Does not accept a left and right margin.
May need a better algorithm for leaving lines unjustified.

Associated files

`justify.exec, justify.s`

See also

format, **dumpscreen**, **roff**, **1616typer**

Distribution

Applix 1616 Shareware Disk #3 /printer

Author

Original **print.s** by Andrew Morton

This version by N R Clark

kal

Description

Sixteen colour kaleidoscope in 320 by 200 mode, produced using ***drawline*** routines. Clears screen and produces new patterns fairly often. Restores previous video characteristics on exit. Cheerful and eye catching.

	Any key starts the display.
Esc	quits to operating system.
p	acts as a toggle to pause the display.

Quibbles

Leaves block cursor in the top left hand corner of display.
Lots of noise out the soundport (as with many programs). This is actually a hardware problem.

Associated files

```
kal.xrel
kal.c
```

See also

life, **frac**

Distribution

Applix 1616 Shareware Disk #1 root directory

Author

Jeremy Fitzhardinge

macshow - display a Macintosh picture file - Andrew McNarara - SW#1

macshow `filename`

Description

Display a picture file from a Macintosh. Earlier version of **macpic** program. Gives "uncrunching data" message during the few seconds it takes processing image file. Only works with Macintosh picture files.

up arrow	Scroll up
down arrow	Scroll down
Page up	Move 4 lines up
Page Dn	Move 4 lines down
Space Bar	Blanks display
I	Inverts image, or negative image
Q	Quit from program

Examples

```
macshow max
```

Quibbles

Picture is distorted due to difference between 512 by 342 Macintosh resolution, and 640 by 200 Applix display. Use **macpic**, which fixes the aspect ratio, and also displays the original image.

Associated files

`macshow.exec`
`macshow.s`
`pictures.doc`
`beatles, max, sting` (sample picture files)
Vast numbers of pictures of [semi] naked women, which is probably prohibited under Justice Einfeld’s interpretation of the Sex Discrimination Act.

See also

macpic

Distribution

Applix 1616 Shareware Disk #1 /pictures

Author

Andrew McNamara

macpic - display a Macintosh picture file - Andrew McNarara - SW#3

macpic `filename ...`

Description

Display a picture file from a Macintosh. Revised, updated replacement for **macshow** program. Gives "uncrunching data" message during the few seconds it takes processing image file. Only works with Macintosh picture files. Now accepts multiple file names, and lets you see next and previous pictures. Rewritten in C, with optimised assembler patches.

up arrow	Scroll up
down arrow	Scroll down
Page up	Move 4 lines up
Page Dn	Move 4 lines down
Space Bar	Blanks display
a	Aspect ratio changed
i	Inverts image, or negative image
n	Next picture file
p	Previous picture
q	Quit from program

Examples

```
macpic max
```

Associated files

`macpic.xrel`
`macpic.c`
`pictures.doc` (which Andrew doesn't recall writing ... Kathy?)
`beatles, max, sting` (sample picture files)
Vast numbers of pictures of [semi] naked women, which is probably prohibited under Justice Einfeld's interpretation of the Sex Discrimination Act.

See also

macshow

Distribution

Applix 1616 Shareware Disk #3 (or later disk?)

Author

Andrew McNamara

mand

Description

Mandelbrot demonstration using 32 bit floating point routine Gerhard developed. Use arrow keys to move and resize the box, p to recalculate area within box.

Same as the version in Forth, in appearance.

Quibbles

No source; that will probably appear on SW#17 or SW#18 ... it arrived later.

Associated files

`mand.exec`

See also

kal

Distribution

Applix 1616 Shareware Disk #14 /baumann

Author

Gerhard Baumann

pic - visual display of play back sounds - Gerhard Baumann

pic `soundfile`

Description

Plays sound files, together with a visual display of the sound wave. The screen clear does not keep up with this program, so the sound is distorted, and playback is not in real time.

Quibbles

Associated files

`isr.as, isr.obj, makefile, soundisr.s`

See also

playsound, back, sound11, etc

Distribution

None as yet, arrived October 1989.

Author

Gerhard Baumann

play - play back sounds - Gerhard Baumann - SW#9

play [`freq`] `soundfile` [`soundfile` ...]

Description

Short program to play back speach and sounds. Looks like the original `play11` in C, modified to accept varying size files, and play back at a specified rate. Will attempt to work on soundfiles too large for memory.

For the frequency, try .53 for 11 KHz, .38 for 15 KHz, .22 for 22 KHz files.

Quibbles

Needs to be modified for 15 MHz 1616 systems.
`Play1` appears to be the same program, but with only one argument, a single soundfile.

Associated files

```
isr.as, isr.obj, makefile, play.c, play.obj, play.xrel,
play1.c, soundisr.s
```

See also

playsound, back, sound11, etc

Distribution

Applix 1616 Shareware Disk #9 /sound

Author

I think this is Gerhard Baumann’s version.

play_s - play back stereo sounds - Gerhard Baumann

play `freq  soundfile`

Description

Short program to play back speach and sounds recorded in stereo. Looks like the original `play11` in C, modified to play stereo through two sound channels. Will attempt to work on soundfiles too large for memory.

For the frequency, try .53 for 11 KHz, .38 for 15 KHz, .22 for 22 KHz files.

Quibbles

Needs to be modified for 15 MHz 1616 systems.
Will play back non-stereo soundfiles, but they will sound distorted.

Associated files

`isr.as, isr.obj, makefile, soundisr.s`

See also

playsound, back, sound11, etc

Distribution

None as yet, arrived October 1989.

Author

Gerhard Baumann

playsound - plays digitised sound files - Andrew Morton - SW#1

playsound `soundfile`

Description

Reads a digitised sound file from disk, and plays it using the ***freetone*** system call. Automates `playsound.shell`.

Examples

```
playsound bogie.snd.11
```

Quibbles

Accepts pretty much any type of file.
You can give it a sound file too large for memory (try `coke.snd.11`).
Should be replaced with new versions on Shareware Disk #3 or SW#6.

Associated files

`playsound.exec`
`playsound.s`
`bogie.snd.11, coke.snd.11, max.snd.11` (sample sound files)

See also

back, back11, sound-11, sound22, sound-22

Distribution

Applix 1616 Shareware Disk #1 /sound

Author

Andrew Morton

reverb - Reverberation on Mac digitised sound files - Jeremy Fitzhardinge - ZZZSW#1, SW#6

reverb `soundfile` ptr1_off ptr1_lvl prt2_off ptr2_lvl ptr2_off ptr3_lvl

Description

Adds reverberation effects to 11 KHz digitised sound files obtained from Macintosh. Revised assembly version of `sound11, back`. Checks to ensure you have Version 3 EPROMS.

Plays the sound with three sets of parameters. *ptr1_off* is the first offset of the reverb from the main sound, and *ptr1_lvl* is the level of the sound compared with the main sound. The levels are fractional powers of 2 (and are simply right shifted).

Examples

```
reverb bogie.snd.11 1000 0 2000 1 3000 2
reverb bogie.snd.11 300 0 400 1 500 2
```

Quibbles

Jeremy says he needs a spelling checker.

Associated files

```
reverb.xrel, reverb.s, *.snd.11
```

See also

sound11, **sound-11**, **sound22**, **sound-22**, **back**

Distribution

Applix 1616 Shareware Disk #6 /soundstuff

Author

Jeremy Fitzhardinge

runoff `[+n] [-n] [-p] [-s]`

Description

The first of the shareware text formatters to be able to display **bold**, *italics*, etc, on the display. The text file must have the appropriate control lines and escape characters embedded in it. The file is printed by using the `imagewriter` program. See the example files `delta.roff` and `runoff.roffin /text`.

+n	Start printing at page *n*.
-n	Stop printing at page *n*.
-p	Send output to printer.
-s	Stop before printing each page, to setup printer.

Control lines begin with a command character, normally full stop `.` The summary below lists the commands, with *n* indicating a numeric value, *t* a title, *c* a single character.

.ad	c	Adjust output lines with mode *c*.
.bl	n	Insert *n* blank lines.
.bp	n	Begin new page, number it *n*, default 1.
.br		Line break.
.c2	c	Set no-break control character to *c*.
.cc	c	Control character becomes *c*.
.ce	n	Centre the next *n* lines.
.ds		Double space, same as .ls 24.
.ec	c	Set escape character to *c* (default is \).
.eo		Toggle excape character mechanism (default on).
.fi		Begin filling output lines (default yes).
.fo	t	All footer titles are *t*, default none.
.he	t	All header titles are *t*, default none.
.in	+n	Indent *n* spaces from left margin.
.ll	+n	Line length, including indent, default 65.
.ls	+n	Line spacing is *n*/72nd inch, default 12.
.lt	+n	Length of title lines, default 65.
.m1	n	*n* blank lines between top of page and header, default 2.

.m2	n	*n* blank lines between header and text, default 2.
.m3	n	*n* blank lines between text and footer, default 2.
.m4	n	*n* blank lines between footer and bottom, default 3.
.na		No output line adjusting.
.ne	n	Begin new page if within *n* of bottom.
.nf		Stop filling output lines.
.os		Print lines sdaved with `.sv n`.
.pc	c	Set page character to *c*.
.pl	+n	Paper length is 12*n*/72 inches, default 70.
.pn	+n	Number next page *n*, default 1.
.po	+n	Page offset, preceed lines by *n* spaces, default 8.
.sk	n	Produce *n* blank pages.
.sp	n	Insert *n* blank lines, except at top.
.ss		Single space, same as `.ls 12`.
.sv	n	Save *n* blank lines if no room.
.ti	+n	Indent next line only *n* spaces.
.tl	t	Print title *t*.

Escape control codes, escape normally a `\`, but can be changed with `.ec`.

\-0	Cancel underline mode.
\-1	Set continuous underline mode.
_0	Cancel undeline mode.
_1	Set non-continuous underline mode. mode.
\4	Set *italic* mode.
\5	Cancel italic mode.
\E	Set **emphasised** mode.
\F	Cancel emphasised mode.
\S0	Set $^{\text{superscript}}$ mode.
\S1	Set $_{\text{subscript}}$ mode.
\T	Cancel subscript and superscript.
\W1	Set enlarged mode.
\W0	Cancel enlarged mode.
\x1	Set letter quality print mode.

\x0	Cancel letter quality print mode.
\' '	Unpaddable space sized character.

Quibbles

Associated files

`imagewriter.exec, runoff.exec, in /bin,`
`/chsets directory contains ibm.chset, normal.chset, old-font.chset, sub-ibm.chset, subscript.chset, sup-ibm.chset,superscript.chset,`
`/runoff directory contains ibmfont.s, imagewriter.s, italicfont.s, printeribm.s, runoff.s, subfont.s, subitalfont.s` 332 blocks

See also

dde

Distribution

Applix 1616 Shareware Disk #10

Author

Craig Mills.

sound-11 - plays Mac digitised sound files (revised) - Matthew Geier - SW#3

sound-11 `soundfile`

Description

Plays 11 KHz digitised sound files obtained from Macintosh. Revised assembly version of `sound11`. Uses ***getmem*** to ensure there is sufficient room in memory. Version 5 is the latest I've seen. If using large sound files, remember to boot with an MRD that sets /RD and stack to smaller sizes.

Examples

```
sound-11 bogie.snd.11
```

Quibbles

Associated files

`sound-11.xrel, sound-11.s, sound.doc, *.snd.11`

See also

sound11, **back**, **sound22**, **sound-22**

Distribution

Applix 1616 Shareware Disk #3 /moresound

Author

Matthew Geier

sound-22 - plays Mac digitised sound files (revised) - Matthew Geier - SW#3

sound-22 `soundfile`

Description

Plays 22 KHz digitised sound files obtained from Macintosh. Revised assembly version of `sound11`. Uses ***getmem*** to ensure there is sufficient room in memory.

Examples

```
sound-22 superman.snd.22
```

Quibbles

Associated files

`sound-22.xrel, sound-22.s, *.snd.22`

See also

sound11, **back**, **sound22**, **sound-11**

Distribution

Applix 1616 Shareware Disk #3 /moresound

Author

Matthew Geier

sound11 - plays digitised sound files - Andrew Morton - SW#1

sound11 `soundfile`

Description

Reads an 11 KHz digitised sound file from disk, and plays it using the ***freetone*** system call. Doesn't work real well if the soundfile is other than 11KHz. Since there are other sound programs, you should carefully identify 11KHz sound files with the extension `.snd.11`.

Examples

```
sound11 bogie.snd.11
```

Quibbles

Accepts pretty much any type of file.
You can give it a sound file too large for memory (try `coke.snd.11`).
Should be replaced with new versions on Shareware Disk #3.

Associated files

`sound11.xrel`
`sound11.s`
`bogie.snd.11, coke.snd.11, max.snd.11` (sample sound files)

See also

back, back11, sound-11, sound22, sound-22

Distribution

Applix 1616 Shareware Disk #1 /sound

Author

Andrew Morton

sound22 - plays Mac digitised sound files (revised) - Matthew Geier - SW#3

sound22 `soundfile`

Description

Plays 22 KHz digitised sound files obtained from Macintosh. Revised C language version of `sound11`. Uses ***getmem*** to ensure there is sufficient room in memory.

Examples

```
sound22 superman.snd.22
```

Quibbles

Associated files

```
sound-22.xrel, sound-22.c, *.snd.22
```

See also

sound11, **back**, **sound22**, **sound-**11

Distribution

Applix 1616 Shareware Disk #3 /moresound

Author

Matthew Geier

soundsloman

Description

Reads in and plays a `.snd` file, uses ***freetone*** to play it.

Quibbles

Associated files

`soundsloman.s, ACDC,dreadlock, radio`

See also

sound, **wave**

Distribution

Applix 1616 Shareware Disk #17 /sFX

Author

Michael J Sloman, 1642 Main Nth Rd, Brahma Lodge SA 5109

speakfile - plays digitised sound files - Andrew Morton - SW#1

speakfile `soundfile`

Description

Reads a digitised sound file from disk, and plays it using the ***freetone*** system call. Automates `playsound.shell`.

Examples

```
speakfile bogie.snd.11
```

Quibbles

Accepts pretty much any type of file.
You can give it a sound file too large for memory (try `coke.snd.11`).
Should be replaced with new versions on Shareware Disk #3 or SW#6.

Associated files

`speakfile.exec`
`speakfile.s`
`bogie.snd.11, coke.snd.11, max.snd.11` (sample sound files)

See also

back, back11, sound-11, sound22, sound-22

Distribution

Applix 1616 Shareware Disk #1 /sound

Author

Andrew Morton

sseg - EGA graphics package - Conal Walsh - SW#12

sseg

Description

EGA mode graphics package, which extends the ***set_640*** syscall to EGA using new modes 12 and 16. Requires an EGA, dual scan or multisync monitor. Do not use with a standard CGA monitor, as you may damage the monitor sync circuitry. See tutorial, starting in μPeripheral #10.

A modular set of 68000 assembler routines, which can be accessed as a C library, or from a memory resident driver. Some of the demos (which are well worth looking at, even if you don't intend to use them) require the `mrdrivers` file, so boot from the floppy.

To install `sseg` for your C compiler,
copy `graphics.h` to your own `include` files directory.
copy `libg.lib` and `libgr.lib` to your `libraries` directory.
In your programs, add `#include graphics.h` at the start of the program, and include either the `-lg` switch to link with the the `sseg` routines, or `-lgr` to interface with the `sseg mrdrivers`.

The demos use a common calling convention for specifying the video mode:
`program` -h   use mode 16 (640x350, 4 colours)
`program` -m   use mode 12 (320x350, 16 colours).
`program` -c   use mode 0 (320x200, 16 colours).

Quibbles

Associated files

```
/assembler/ sseg.newlib, sslib.macros, ssres.macros,
ssres.newlib
/assembler/demos (12 files)
/assembler/test (6 files)
/c/ cres.as, libg.lib, libgr.lib
/c/demos (14 files)
/c/include/graphics.h
/source (40 files)
/mrdrivers
/sseg.mrd
```

See also

c1616 help file.

Distribution

Applix 1616 Shareware Disk #12 786 blocks

Author

Conal Walsh.

termite

Description

Short program to eat a display, while producing noisy munching on any visible material.

Quibbles

No source file with this?

Associated files

`termite.doc, termite.xrel`

See also

disolve

Distribution

Applix 1616 Shareware Disk #8

Author

Andrew McNamara.

tiff - tag sound format files - Lindsay Scales - SW#10

tiff description

Description

Sound files can be stored as a continuous stream of absolute value data bytes, however as there is little variation from sample to sample, they can be compressed by recording only the difference between successive bytes.

Tag Sound Format Files add a variety of parameters to a file, to allow the identification of a wide variety of files, including mixed image, sound and text data.

Quibbles

Associated files

`music.tags, tiff.head, tsff.doc`

See also

Distribution

Applix 1616 Shareware Disk #10 /tiff

Author

Probably Lindsay Scales.

tune3 `musicfile`

Description

Tune3 plays musical notes, according to a list in a music file. The macro capabilities of the `ssasm` assembler are used to turn a text file into a format suitable for playing using `tune3`. The text files are normally given the extension `.mus`, but must have a `.s` extension prior to being assembled. The resulting `.exec` file is renamed to get rid of the `.exec` extension prior to being played.

`copy roncalli.mus yourfile.s`
(This makes a copy in assembler format)
`edit yourfile.s`
(Modify to the tune you require)
`ssasm yourfile`
(Assemble the file, including macros)
`rename yourfile.exec yourfile`
(Get rid of the `exec` suffix)
`tune3 yourfile`
(Play the tune)

Now you will need to know just what you should do when editing. The keywords used in editing music files are as follows:

TEMPO	Specify a number for the slowness of the piece, e.g.3000.
DECAY	Part#, value. part# is 1,2 or 3. Value is 1 (long) to 4 (short).
VOLUME	Part#, value. Values can range from 0 (off) to 255 (full).
CHORD	Duration, Pitch1, Pitch2, Pitch3 Duration is 1 (quave), 2 (crotchet), 4 (minim), 8 (semibreve) etc. Pitch is 1 (C below bass clef), and increasing in semitones to about 50.

Modifications (notes by Lindsay Scales)

To make the notes easier to write, and more readable, make these simple modifications to provide note names (these are a subset of the MIDI range). Edit `compose.mac` to add the contents of `musicdefs.mac` and `compose2.mac`. Then add the following equates, where:

N specifies a musical note (to prevent confusion with address and data register equates).

C is the note name (A, B, C, D, E, F, G).

F or **S** is added to indicate sharps or flats.

3, 4, 5, 6 is the octave number, spanning from A flat to G sharp.

eg.

```
NC3     EQU    1     Lowest note in Tune3
NCS3    EQU    2     Provide both a sharp
NDF3    EQU    2     and flat name for note
ND3     EQU    3
...     ...    ...   more notes
NGS3    EQU    9     Octave changes after G sharp
NAF4    EQU    9     Another octave, and repeat ...
...     ...    ...
```

Finally, change the *Include* statements in the `.s` (`.mus`) file to: `include compose.mac`. Other equates could also be used, such as `long equ 1` for note decays, `crotchet equ 2` for note values, and `ff equ 255` for note loudness.

This system of macro expansions would be quite valuable for other applications, where data files need to be knocked up quickly, such as MIDI data, bit mapped graphics file headers, setup files for printers, etc.

Quibbles

Andrew didn't document this.

Associated files

`ssasm.xrel, tune3.xrel, musicdefs.mac, compose2.mac`
Music files by Andrew: `roncalli, bach`
Music files by Craig Mills: `duel, green, maid, saints, short, stairway, wedding` on SW#9 /mills

See also

sound, play, etc.

Distribution

`Tune3` on various User Disks.
Music files by Craig Mills on SW#9 /mills

Author

Andrew Morton.

vi - powerful Minix text editor - Colin McCormack, Andrew Morton - SW#9

vi

Description

Unix editor.

Quibbles

Associated files

See also

42 files in directory!

Distribution

Applix 1616 Shareware Disk #9 /vi

Author

Conversion by Colin McCormack and Andrew Morton

STEVIE - An Aspiring VI Clone

User Reference - 3.44, by Tony Andrews

Overview

STEVIE is an editor designed to mimic the interface of the UNIX editor ‘vi’. The name (ST Editor for VI Enthusiasts) comes from the fact that the editor was first written for the Atari ST. The current version has been ported to UNIX, Minix, MS-DOS, and OS/2, but I’ve left the name intact for now.

This program is the result of many late nights of hacking over the last year or so. The first version was written by Tim Thompson and posted to USENET. From there, I reworked the data structures completely, added LOTS of features, and generally improved the overall performance in the process.

I’ve labelled STEVIE an ‘aspiring’ vi clone as a warning to those who may expect too much. On the whole, the editor is pretty complete. Almost all of the visual mode commands are supported. I’ve tried very hard to capture the ‘feel’ of vi by getting the little things right. Making lines wrap correctly, supporting true operators, and even getting the cursor to land on the right place for tabs are all a real pain, but really help make the editor ‘feel’ right.

STEVIE may be freely distributed. The source isn’t copyrighted or restricted in any way. If you pass the program along, please include all the documentation and, if practical, the source as well. I’m not fanatical about this, but I tried to make STEVIE fairly portable and that doesn’t do any good if the source isn’t available.

The remainder of this document describes the operation of the editor. This is intended as a reference for users already familiar with the real vi editor.

Starting the Editor

The following command line forms are supported:

`stevie [file ...]`	Edit the specified file(s)
`stevie -t tag`	Start at the location of the given tag
`stevie + file`	Edit file starting at end
`stevie +n file`	Edit file starting a line number ‘n’
`stevie +/pat file`	Edit file starting at pattern ‘pat’

If multiple files are given on the command line (using the first form), the `:n` command goes to the next file, `:N` goes backward in the list, and `:rew` can be used to rewind back to the start of the file list.

Set Command Options

The `:set` command works as usual to set parameters. Each parameter has a long and an abbreviated name, either of which may be used. Boolean parameters are set as in:

```
set showmatch
```

or cleared by:

```
set noshowmatch
```

Numeric parameters are set as in:

```
set scroll=5
```

Several parameters may be set with a single command:

```
set novb sm report=1
```

To see the status of all parameters use `:set all`. Typing `:set` with no arguments will show only those parameters that have been changed. The supported parameters, their names, abbreviations, defaults, and descriptions are shown below:

autoindent
Short: ai, Default: noai, Type: Boolean
When in insert mode, start new lines at the same column as the prior line.
Unlike vi, you can backspace over the indentation.

backup
Short: bk, Default: nobk, Type: Boolean
Leave a backup on file writes.

errorbells
Short: eb, Default: noeb, Type: Boolean
Ring bell when error messages are shown.

ignorecase
Short: ic, Default: noic, Type: Boolean
Ignore case in string searches.

lines
Short: lines, Default: lines=25, Type: Numeric
Number of physical lines on the screen. The default value actually depends on the host machine, but is generally 25.

list
Short: list, Default: nolist, Type: Boolean
Show tabs and newlines graphically.

number
Short: nu, Default: nonu, Type: Boolean
Display lines on the screen with their line numbers.

report
Short: report, Default: report=5, Type: Numeric
Minimum number of lines to report operations on.

return
Short: cr, Default: cr, Type: Boolean
End lines with cr-lf when writing files.

scroll
Short: scroll, Default: scroll=12, Type: Numeric
Number of lines to scroll for ^D & ^U.

showmatch
Short: sm, Default: nosm, Type: Boolean
When a), }, or] is typed, show the matching (, {, or [if it's on the current screen by moving the cursor there briefly.

showmode
Short: mo, Default: nomo, Type: Boolean
Show on status line when in insert mode.

tabstop
Short: ts, Default: ts=8, Type: Numeric
Number of spaces in a tab.

wrapscan
Short: ws, Default: ws, Type: Boolean
String searches wrap around the ends of the file.

vbell
Short: vb, Default: vb, Type: Boolean
Use a visual bell, if possible. (novb for audible bell)

The EXINIT environment variable can be used to modify the default values on startup as in:

```
setenv EXINIT="set sm ts=4"
```

The 'backup' parameter, if set, causes the editor to retain a backup of any files that are written. During file writes, a backup is always kept for safety until the write is completed. At that point, the 'backup' parameter determines whether the backup file is deleted.

In environments (e.g. OS/2 or TOS) where lines are normally terminated by CR-LF, the 'return' parameter allows files to be written with only a LF terminator (if the parameter is cleared). This parameter is ignored on UNIX systems.

The 'lines' parameter tells the editor how many lines there are on the screen. This is useful on systems like the ST (or OS/2 machines with an EGA adapter) where various screen resolutions may be used. By using the 'lines' parameter, different screen sizes can be easily handled.

: Colon Commands

Several of the normal ‘vi’ colon commands are supported by STEVIE. Some commands may be preceded by a line range specification. For commands that accept a range of lines, the following address forms are supported:

```
addr addr + number addr - number
```

where ‘addr’ may be one of the following:

a line number
a mark (as in ’a or ’b)
’.’ (the current line)
’$’ (the last line)

An address range of "%" is accepted as an abbreviation of "1,$".

The Global Commands

A limited form of the global command is supported, accepting the following command form:

```
g/pattern/X
```

where X may be either ‘d’ or ‘p’ to delete or print lines that match the given pattern. If a line range is given, only those lines are checked for a match with the pattern. If no range is given, all lines are checked.

If the trailing command character is omitted, ‘p’ is assumed. In this case, the trailing slash is also optional. The current version of the editor does not support the undo operation following the deletion of lines with the global command.

The Substitute Command

The substitute command provides a powerful mechanism for making more complex substitutions than can be done directly from visual mode. The general form of the command is:

```
s/pattern/replacement/g
```

Each line in the given range (or the current line, if no range was given) is scanned for the given regular expression. When found, the string that matched the pattern is replaced with the given replacement string. If the replacement string is null, each matching pattern string is deleted.

The trailing ‘g’ is optional and, if present, indicates that multiple occurrences of ‘pattern’ on a line should all be replaced.

Some special sequences are recognized in the replacement string. The ampersand character is replaced by the entire pattern that was matched. For example, the following command could be used to put all occurrences of ‘foo’ or ‘bar’ within double quotes:

```
1,$s/foo|bar/"&"/g
```

The special sequence "\n" where ‘n’ is a digit from 1 to 9, is replaced by the string that matched the corresponding parenthesized expression in the pattern. The following command could be used to swap the first two parameters in calls to the C function "foo":

```
1,$s/foo\(([^,]*),([^,]*),/foo(\2,\1,/g
```

Like the global command, substitutions can’t be undone with this version of the editor.

File Manipulation Commands

The following table shows the supported file manipulation commands as well as some other ‘ex’ commands that aren’t described elsewhere:

:w	write the current file
:wq	write and quit
:x	write (if necessary) and quit
ZZ	same as ":x"
:e file	edit the named file
:e!	re-edit the current file, discarding changes
:e #	edit the alternate file
:w file	write the buffer to the named file
:x,yw file	write lines x through y to the named file
:r file	read the named file into the buffer
:n	edit the next file
:N	edit the previous file
:rew	rewind the file list
:f	show the current file name
:f name	change the current file name
:x=	show the line number of address ‘x’
:ta tag	go to the named tag
^]	like ":ta" using the current word as the tag
:help	display a command summary
:ve	show the version number
:sh	run an interactive shell
:!cmd	run a command

The ":help" command can also be invoked with the <HELP> key on the Atari ST. This actually displays a pretty complete summary of the real vi with unsupported features indicated appropriately.

The commands above work pretty much like they do in ‘vi’. Most of the commands support a ‘!’ suffix (if appropriate) to discard any pending changes.

String Searches

String searches are supported, as in vi, accepting the usual regular expression syntax. This was done using a modified form of Henry Spencer’s regular expression library. I added code outside the library to support the ‘\<’ and ‘\>’ extensions. This actually turned out to be pretty easy, although there may be some glitches in the way I did it. The parameter "ignorecase" can be set to ignore case in all string searches.

Operators

The vi operators (d, c, y, !, <, and >) work as true operators. The only exception is that the change operator works only for character-oriented changes (like cw or c%) and not for line-oriented changes (like cL or c3j).

Tags

Tags are implemented and a fairly simple version of ‘ctags’ is supplied with the editor. The current version of ctags will find functions and macros following a specific (but common) form. See ‘ctags.doc’ for a complete discussion.

System Specific Comments

The following sections provide additional relevant information for the systems to which STEVIE has been ported.

Atari ST

TOS. The editor has been tested in all three resolutions, although low and high res. are less tested than medium. The 50-line high res. mode can be used by setting the ‘lines’ parameter to 50. Alternatively, the environment variable ‘LINES’ can be set. The editor doesn’t actively set the number of lines on the screen. It just operates using the number of lines it was told.

The arrow keys, as well as the <INSERT>, <HELP>, and <UNDO> keys are all mapped appropriately.

Minix

The editor is pretty much the same under Minix, but many of the keyboard mappings aren’t supported.

UNIX

The editor has been ported to UNIX System V release 3 as well as 4.2 BSD. This was done mainly to get some profiling data so I haven't put much effort into doing the UNIX version right. It's hard-coded for ansi-style escape sequences and doesn't use the termcap/terminfo routines at all.

OS/2

This port was done because the editor that comes with the OS/2 developer's kit really stinks. Make sure 'ansi' mode is on (using the 'ansi' command). The OS/2 console driver doesn't support insert/delete line, so STEVIE bypasses the driver and makes the appropriate system calls directly. This is all done in the system-specific part of the editor so the kludge is at least localized.

The arrow keys, page up/down and home/end all do what you'd expect. The function keys are hard-coded to some useful macros until I can get true support for macros into the editor. The current mappings are:

F1 :p <RETURN>
F2 :n <RETURN>
F3 :e # <RETURN>
F4 :rew <RETURN>
F5 [[F6]]
F7 << F8 >>
F9 :x <RETURN>
F10 :help <RETURN>

S-F1 :p! <RETURN>
S-F2 :n! <RETURN>

MS-DOS

STEVIE has been ported to MSDOS 3.3 using the Microsoft C compiler, version 5.1. The keyboard mappings are the same as for OS/2. The only problem with the PC version is that the inefficiency of the screen update code becomes painfully apparent on slower machines.

Missing Features

1. Counts aren't yet supported everywhere that they should be.
2. Macros with support for function keys.
3. More "set" options.
4. Many others...

Known Bugs and Problems

1. The change operator is only half-way implemented. It works for character motions but not line motions. This isn’t so bad since most change operations are character oriented anyway.

2. The yank buffer uses statically allocated memory, so large yanks will fail. If a delete spans an area larger than the yank buffer, the program asks for confirmation before proceeding. That way, if you were moving text, you don’t get screwed by the limited yank buffer. You just have to move smaller chunks at a time. All the internal buffers (yank, redo, etc.) need to be reworked to allocate memory dynamically. The ‘undo’ buffer is now dynamically allocated, so any change can be undone.

3. If you stay in insert mode for a long time, the insert buffer can overflow. The editor will print a message and dump you back into command mode.

4. The current version of the substitute command (i.e. ":s/foo/bar") can’t be undone.

5. Several other less bothersome glitches...

Conclusion

The editor has reached a pretty stable state, and performs well on the systems I use it on, so I’m pretty much in maintenance mode now. There’s still plenty to be done; the screen update code is still pretty inefficient and the yank/put code is still primitive. But after more than a year of hacking, I’m ready to work on new projects. I’m still interested in bug reports, and I do still add a new feature from time to time, but the rate of change is way down now.

I’d like to thank Tim Thompson for writing the original version of the editor. His program was well structured and quite readable. Thanks for giving me a good base to work with.

If you’re reading this file, but didn’t get the source code for STEVIE, it can be had by sending a disk with return postage to the address given below. I can write disks for the Atari ST (SS or DS) or MSDOS (360K or 1.2M). Please be sure to include the return postage. I don’t intend to make money from this program, but I don’t want to lose any either.

I’m not planning to try to coordinate the various ports of STEVIE that may occur. I just don’t have the time. But if you do port it, I’d be interested in hearing about it.

Tony Andrews
UUCP: onecom!wldrdg!tony
5902E Gunbarrel Ave. Boulder, CO 80301

This port by Andrew Morton, Applix Pty Ltd, from Colin McCormack’s Minix version for the Applix 1616. Manual laser printed by Eric Lindsay. I’ll update and correct this asap.

Character Function Summary

The following list describes the meaning of each character that's used by the editor. In some cases characters have meaning in both command and insert mode; these are all described.

^@	The null character. Not used in any mode. This character may not be present in the file, as is the case with vi.
^B	Backward one screen.
^D	Scroll the window down one half screen.
^E	Scroll the screen up one line.
^F	Forward one screen.
^G	Same as ":f" command. Displays file information.
^H	(Backspace) Moves cursor left one space in command mode. In insert mode, erases the last character typed.
^J	Move the cursor down one line.
^L	Clear and redraw the screen.
^M	(Carriage return) Move to the first non-white character in the next line. In insert mode, a carriage return opens a new line for input.
^N	Move the cursor down a line.
^P	Move the cursor up a line.
^U	Scroll the window up one half screen.
^Y	Scroll the screen down one line.
^[	Escape cancels a pending command in command mode, and is used to terminate insert mode.
^]	Moves to the tag whose name is given by the word in which the cursor resides.
^‘	Same as ":e #" if supported (system-dependent).
SPACE	Move the cursor right one column.
!	The filter operator always operates on a range of lines, passing the lines as input to a program, and replacing them with the output of the program. The shorthand command "!!" can be used to filter a number of lines (specified by a preceding count). The command "!" is replaced by the last command used, so "!!!<RETURN>" runs the given number of lines through the last specified command.
$	Move to the end of the current line.
%	If the cursor rests on a paren ‘()’, brace ‘{}’, or bracket ‘[]’, move to the matching one.

’	Used to move the cursor to a previously marked position, as in ’a or ’b. The cursor moves to the start of the marked line. The special mark ’’ refers to the "previous context".
+	Same as carriage return, in command mode.
,	Reverse of the last t, T, f, or F command.
-	Move to the first non-white character in the previous line.
.	Repeat the last edit command.
/	Start of a forward string search command. String searches may be optionally terminated with a closing slash. To search for a slash use ‘\/’ in the search string.
0	Move to the start of the current line. Also used within counts.
1-9	Used to add ‘count’ prefixes to commands.
:	Prefix character for "ex" commands.
;	Repeat last t, T, f, or F command.
<	The ‘left shift’ operator.
>	The ‘right shift’ operator.
?	Same as ‘/’, but search backward.
A	Append at the end of the current line.
B	Backward one blank-delimited word.
C	Change the rest of the current line.
D	Delete the rest of the current line.
E	End of the end of a blank-delimited word.
F	Find a character backward on the current line.
G	Go to the given line number (end of file, by default).
H	Move to the first non-white char. on the top screen line.
I	Insert before the first non-white char. on the current line.
J	Join two lines.
L	Move to the first non-white char. on the bottom screen line.
M	Move to the first non-white char. on the middle screen line.
N	Reverse the last string search.
O	Open a new line above the current line, and start inserting.
P	Put the yank/delete buffer before the current cursor position.

R	Replace characters until an "escape" character is received. Similar to insert mode, but replaces instead of inserting. Typing a newline in replace mode is the same as in insert mode, but replacing continues on the new line.
T	Reverse search ‘upto’ the given character.
U	Restore the current line to its state before you started changing it.
W	Move forward one blank-delimited word.
X	Delete one character before the cursor.
Y	Yank the current line. Same as ‘yy’.
ZZ	Exit from the editor, saving changes if necessary.
[[	Move backward one C function.
]]	Move forward one C function.
^	Move to the first non-white on the current line.
‘	Move to the given mark, as with ’. The distinction between the two commands is important when used with operators. I support the difference correctly. If you don’t know what I’m talking about, don’t worry, it won’t matter to you.
a	Append text after the cursor.
b	Back one word.
c	The change operator.
d	The delete operator.
e	Move to the end of a word.
f	Find a character on the current line.
h	Move left one column.
i	Insert text before the cursor.
j	Move down one line.
k	Move up one line.
l	Move right one column.
m	Set a mark at the current position (e.g. ma or mb).
n	Repeat the last string search.
o	Open a new line and start inserting text.
p	Put the yank/delete buffer after the cursor.
r	Replace a character.
s	Replace characters.
t	Move forward ‘upto’ the given character on the current line.

u	Undo the last edit.
w	Move forward one word.
x	Delete the character under the cursor.
y	The yank operator.
z	Redraw the screen with the current line at the top (zRETURN), the middle (z.), or the bottom (z-).
|	Move to the column given by the preceding count.

video - MRD to provide EGA and MicroBee video formats - Conal Walsh - SW#6

video Needs to be installed as an MRD on boot.

Description

A full set of files, to provide two sets of custom video drivers. You will need the HiTech compiler to recompile your own versions, however there are two complete MRD's here, ready to install.

EGA provides resolution of 640 by 350 on either a Hercules monochrome display, or on a multisync colour display.

MBV supplies 512 by 256, for emulating the MicroBee 16 by 8 bit characters, for assistance in porting MicroBee software.

Modified ***set_640*** syscall is used to switch video modes. Normal CGA 320 and 640 modes are provided with parameters of 0 or 1, while the new MBV uses a parameter of 10, and EGA a parameter of 16.

Quibbles

Associated files

Ready to use MRDs are `ega.mrd.prim`, and `mbv.mrd.prim`.
Example shell to make new MRD is `asmv.shell`.
`syscalls.newlib`, `video.global`, `info.video`

See also

sega

Distribution

Applix 1616 Shareware Disk # 6/video

Author

Conal Walsh

video spugm - monsterous changes to video - Jeremy Fitzhardinge, Andrew McNamara - SW#16

various

Description

Various short routines devised during dull moments at Spun Out Users Group Meeting in August 1989. Most go directly to the video ram, and alter things to produce effects to send you blind.

Start one in background, then run almost any video display (such as `kal`).

Quibbles

Associated files

`colours.shell, it.shell, m.s, m.xrel, old,.s, qs.s, qs.xrel, r.s, r.s.1, r.xrel, rr.s, rr.xrel, video1.s, video1.xrel` (nice informative names - read the source and guess what they do).

See also

17 files in directory!

Distribution

Applix 1616 Shareware Disk #16 /SpunOutUsersGroupMeeting

Author

Jeremy Fitzhardinge and Andrew McNamara

wave - displays a digitised sound file - Michael Sloman - SW#6

wave `soundfile`

Description

Displays digitised sound files. Display shows sound pattern, indicates starting and ending address of the portion showing on screen. Allows you to move through the sound file in one direction. Can also play the portion on screen, but the result sounded very strange, regardless of the preload used.

/	next page
[	decrement preload
]	increment preload
e	exit program
p	play sound

The 3201 byte sound samples on SW#17 were made with Michael's A/D converter, and can be played using `wave`.

Examples

```
wave bogie.snd.11
```

Quibbles

Associated files

`wave.exec, wave.s *.snd.11`
on SW#17 `upscale, warning, phaser, phaser2, ph\bird, scrath`

See also

sound11, **sound-11**, **sound22**, **sound-22**, **back**, **reverb**

Distribution

Applix 1616 Shareware Disk #6 /soundstuff
Applix 1616 Shareware Disk #17 (may be different version)

Author

Michael Sloman, 1642 Main Nth Road, Brahma Lodge SA 5109

window - demonstrate window code - Colin McCormack - SW#3

test

Description

Draws pretty pictures of windows, includes source code to initialise, create, display, remove, and move windows, change their border, and select them for output. Nice start for a windowing package.

Quibbles

Associated files

`test.c, wbox.c, wcreate.c, wdump.c, whide.c, wind.c, window.c, window.h, window.i, window.lib, winit.c, wmove.c, wunhide.c`

See also

Conal Walsh’s work

Distribution

Applix 1616 Shareware Disk #3 /window

Author

Colin McCormack

window - demonstration of single window - Sid Young - SW#6

window

Description

Short program to demonstrate a window.

Quibbles

Associated files

`window.s, window.exec`

See also

window

Distribution

Applix 1616 Shareware Disk #6 /tutorial

Author

Sid Young.

Summary

60hzcrtc

640circle

1616typer.bas

adc11 `tablesize soundfile`
adc15 `tablesize soundfile`
adc22 `tablesize soundfile`
adc_s `tablesize soundfile`

adc11 `soundfile`

back22 `soundfile`

back `soundfile`

back11 `soundfile`

clrwin

colour-bars

convert `filein fileout`

crtsav [`time`]

dissolve

doc-write

dumpbro

dumpch

dumpscreen

emacs

fft

format `filename` [*n*]

frac `datafile`

ft etc

imagewriter `filename.roff`

invertch

justify `filename`

kal

macshow `filename`

macpic `filename ...`

mand

pic `soundfile`

play [`freq`] `soundfile` [`soundfile` ...]

play `freq  soundfile`

playsound `soundfile`

reverb `soundfile` ptr1_off ptr1_lvl prt2_off ptr2_lvl ptr2_off ptr3_lvl

runoff [`+n`] [`-n`] [`-p`] [`-s`]

sound-11 `soundfile`

sound-22 `soundfile`

sound11 `soundfile`

sound22 `soundfile`

soundsloman

speakfile `soundfile`

sseg

termite

tiff description

vi

video Needs to be installed as an MRD on boot.

wave `soundfile`

test

window

Table of Contents