Shareware Utilities # 1 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

Programs (where applicable) © Copyright 1989 by the authors. All Rights Reserved.
Manual © Copyright 1989 Eric Lindsay
ISBN 0 947341 ?? ?
MC68000® is a trademark of Motorola Inc.
UNIX® is a trademark of AT&T

A Note to Authors

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 17 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.

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. Special thanks are due here to Michael Johnson, Conal Walsh and Greyham Stoney, who have provided very extensive documentation of their work. Michael's documentation is included here. Greyham's disk utility documentation, being almost as long as this manual, is provided as a separate manual. Conal's tutorials are being (gradually) published in the newsletter.

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 & Print
- 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 is easiest for you.

Eric Lindsay, 6 Hillcrest Ave, Faulconbridge NSW 2776
BH (02) 2189651 Tuesday to Friday Weekends (047) 512258

arc - compress and archive files - Andrew Morton

arc [**-**]{amufdxeplvtc} [biswn] [g*password*] *archive* [*filename ...*]

Description

arc archives, compresses and extracts files. Often used to collect related files into a single file prior to transmission electronically.

a add files to archive.

m move files to archive.

u update files in archive.

f freshen date of files in archive.

d delete files from archive.

x extract files from archive.

e extract files from archive.

p copy archived files to standard output.

l list files in archive.

v verbose listing of files in archive.

t test archive integrity.

c convert entry to new packing method.

b retain backup copy of archive.

i maintain IBM PC compatible archive.

s suppress compression (store only).

w suppress warning messages.

n suppress notes and comments.

g encrypt decrypt archive entry.

Associated files

No source code sighted.

Distribution

unknown

Author

Ported by Andrew Morton?

asciicalc - ascii string evaluation tutorial - Sid Young - SW#6

asciicalc

Description

A demonstration of a possible way to write an ascii string evaluator, uses binary mathematics. Handy code fragment.

Associated files

`asciicalc.exec`, `asciicalc.s`,

Distribution

Applix 1616 Shareware Disk # 6/utilities/sid

Author

Sid Young

basic - Tiny basic - Andrew Morton - SW#14

basic

Description

Gordon Brandly's *Tiny BASIC*, derived from the version published in *Dr Dobbs*, May 1976. This is the 1984 68000 version for the Motorola MEX68KECB, ported to the 1616.

Uses 26 single letter variables only, but accepts everything as either upper or lower case. Reserved words can be used in full, or you can truncate them, and end them with a **.** full stop.
: acts as a separator.

Commands include:
list, load, new, run, save
Expression evaluation can handle:
>= <> = <= < > and perhaps logical expressions.
Reserved words include
next, if, goto, gosub, return, rem, for, step, input, print, poke, stop, call, peek, abs, to, step.
Rnd(*n*) returns an integer between 1 and *n*.
Size returns memory free.
Call allows use of all registers, and uses RTS to return.
Let is optional.
For more details, read the code!

Internal line format appears to be **:** line number as 4 hex digits, rest of line in ASCII. The whole program ends when a **@** is found.

Examples

Quibbles

Associated files

`basic.exec, basic.s`

See also

ssbasic

Distribution

Applix 1616 Shareware Disk #14

Author

Original from Palo Alto Tiny BASIC, in Dr Dobbs, May 1976. 68000 port by Gordon Bradley, Applix port by Andrew Morton

bigbuf - set buffer size for character devices - Andrew Morton - SW#14

```
bigbuf [bufsize] [satx] [sarx] [sbtx] [sbrx] [cent] [kb]
```

Description

A program that lets you more easily set the size of the character buffer for any of the I/O devices. The default buffer size is set to $4000 (16k bytes), while the minimum allowed is 32 characters.

Of most use when running a serial port in background, or for making up a defacto printer buffer.

Examples

`bigbuf 1024 kb` sets keyboard buffer to 1k.
`bigbuf 4096 cent` sets printer buffer to 4k.

Bugs

Associated files

`bigbuf.c`

See also

Distribution

Users Disk V4.0b /source
Applix 1616 Shareware Disk #14 /

Author

Andrew Morton.

bmedit - bit map editor - Andrew McNamara - SW#15

bmedit `bmfile`

Description

Edit 16 by 16 graphic bit maps. The output file is a sequential list of the bitmaps, as 16 longs.

The enlarged bitmap is displayed to the left of the screen, the normal size version to the right of it. A full display of previous bitmaps is shown below.

Use the arrow keys (or `s e x d` keys) to move.

PgUp and PgDn keys (or `r c`) to move to the next or previous bitmap.

Home and End to move to front or back of file.

Space (or `5`) to toggle the dot on or off.

! Swap saved bitmap with current one.

^ Push bitmap into save space.

***** Clear current bitmap.

Del Delete current bitmap.

Ins Insert a bitmap.

q Quit.

s Save and quit.

w Write (save) and quit.

Associated files

`bmedit.c`, `bmedit.xrel`, `bmedit.use`

Distribution

Applix 1616 Shareware Disk # 15 /McNamara

Author

Andrew McNamara

c1616 - help system - Conal Walsh - SW#13

c1616 `helpfile` [`swapfile`]

Description

This is a full feature help system, that is intended (eventually) to replace printed manuals. While not yet complete, it already has a multitude of features, and Conal continually improves it. The next version, for example, will probably be in full MRD form, so that it works as a pop-up. The version of `c1616.xrel` on the disk uses the standalone SSEG library, so you do not need the `sseg.mrd` to use it.

The `helpfile` text displayed by the help system is contained in a `.csh` help format file. If no directory is specified, then the `/help` directory will be searched. You should `assign` this to whatever directory contains the `.csh` format files.

The sample help files on the disk are `hcgrap.csh`, which is the `SSEG!` manual, and `hcprog.csh`, which is the 1616/OS *Programmer's Reference Manual*.

The swapfile **must** be a high speed disk, and is typically the ram disk `/rd`. About 20k space is required, and the program will normally default to `/rd`.

The `.rch` files are the raw text files, containing the definitions and index references, as well as the help information. Use the `convhc.xrel` program to convert a file from this format to the `.csh` format used by `c1616`. The raw help file `.rch` format has more meaning when viewed in the document mode of *Wordstar 4* on `Zrdos`, but should also display correctly under the `DrDoc` editor from Applix.

Conal reports that the *Programmer's Manual* took 35 hours to convert, so he is not inclined to do more manuals in a hurry. This means we need people to convert manuals. ASCII versions of all the Applix manuals are available on disk from Eric, **if** you are willing to convert a manual to Conal's `.rch` and `.csh` format.

The `.rch` format consists of a text file, with the key words and phrases surrounded by different control characters. The current version of `c1616` is set to accept pages up to 15 lines long, with up to 45 characters per line. Any page exceeding this will generate a warning from `convhc.xrel`, and will not display correctly. Therefore, the raw text file must be set up in these small pages.

^B*word*^B — *word* is a definition item: when this item is referenced elsewhere, this is the page that will be displayed.

^S*word*^S — *word* is a reference item: it must be defined somewhere else in the document. There can be any number of references to a single definition item.

^D*word*^D *word* will be emphasised when it is displayed using `c1616`; it has no other special meaning.

^L This marks the end of a page.

The Del character (ASCII 127, or 7F) marks the end of a topic; these determine whether the user can use the PgUp and PgDn keys to step forward or backward.

Example

Type `c1616 ./hcgrap` while in directory `/f0/C1616`.
Use ? to get a command summary.
If you have trouble starting it, make the following `assigns`.

```
assign /help .
assign /rd .
```

Associated files

`-readme, auto.shell, c1616.as, c1616.xrel, convhc.c, conchc.xrel, hcgrap.csh, hcgrap.rch, hcprog.csh, hcprog.rch`

Distribution

Applix 1616 Shareware Disk # 13 /c1616

Author

Conal Walsh.

calendar - prints whole year calendar - Paul Cahill - SW#4

calendar.bas

Description

Prints a whole year calendar, for a year from 1900 to 1999, one month after another.

Uses the low resolution display, and returns display to 640 mode when done.

a	Another calendar.
Enter	End program.

Quibbles

Associated files

`calendar.bas`

See also

cal (any year, month)

Distribution

Applix 1616 Shareware Disk #4 /utilities

Author

Paul Cahill

cdev - dump character device tables - Andrew McNamara - SW#15

cdev [device:]

Description

Dumps table of information associated with a character oriented input or output device, showing input and output as separate items. You can optionally include a device name, and may get additional information on that device.

The devices are **con: sa: sb: cent: null:**

Table includes:

```
fd   cdev name    doio    status     passval    lastline
```

Examples

```
cdev con:
```
(lists console statistics, also shows contents of the last line recall buffers.

Quibbles

Associated files

`cdev.c, cdev.xrel` (v2, 27-8-1989)

See also

Distribution

Applix 1616 Shareware Disk #15 /McNamara

Author

Andrew McNamara

dateset - set system clock - David Fowler - SW#4

dateset.bas

Description

Prompts for input of the date and time, in decimal.
Sets the system clock, using the setdate command. Helpful if you can't recall the normal command parameters

Quibbles

Now that the `help` command gives full details, I'm not sure any aid is required.

Associated files

`dateset.bas`

See also

sdate

Distribution

Applix 1616 Shareware Disk # 4/fowler

Author

David A Fowler, P. O. Box 269, Coffs Harbour Jetty, NSW 2450

dateset - at power up, by Michael Johnson - SW#5

author:- MICHAEL JOHNSON, WAGGA WAGGA
program source code:- DATESET.S
date:- 15-9-88
version :- 1.1
document issue :- 2

INTRODUCTION

This program is used to set the system date and time. It designed for use at power up only.

USAGE

This program is designed to be called directly from your boot code at a level 0 reset. It uses a specific area on the screen for its prompt string. The current setting for each part of the date is displayed at the appropriate time with the cursor directly after it. The input will only display 2 characters and each time you press a key the numbers are shifted left and the new 2 digit number is displayed. When you are satisfied that the displayed number is correct you hit the ENTER key and the number is stored. The cursor then steps to the next part to be set and prints it's current contents. The above procedure is then repeated.

If you have made an error then the backspace key will store the current part and step back to the previous part.

When you hit ENTER after the seconds have been input, then the program will store the new values as the current system time and date.

KIND REGARDS

MICHAEL JOHNSON

11/12 KOKODA ST. WAGGA WAGGA. 2650

(069) 255255

debug - breakpoint debugger - Gerhard Baumann - SW#14

Description

The debugger displays the data and address registers, together with the program counter, the contents of the address pointed to by the program counter, the mnemonic of the instruction, and the status register.

Include the file `z.s` in the source program, then invoke the macros `trint`, then `trace`, which enable tracing from that point on.

s Enable automatic stepping, disables other functions. Pressing it again enables manual stepping.

g Continue until next breakpoint.

o Escape from `debug` to 1616/OS temporarily (type `quit` to return).

q Quit the debug program.

r Changes contents of address or data register.

Enter Executes opcode currently being displayed.

Examples

Quibbles

Associated files

`debug.xrel, debug.doc, debug.s,`

See also

disassembler

Distribution

Applix 1616 Shareware Disk #14 /baumann/debug

Author

Gerhard Baumann.

demonstration - of exception handler by Michael Johnson - SW#5

author	:- MICHAEL JOHNSON, WAGGA WAGGA
program name	:- DEMONSTRATION.XREL
program source code	:- DEMONSTRATION.S
date	:- 14-9-88
version	:- 2.0
document issue	:- 2

INTRODUCTION

This program is supplied as an example of how to use the LOCATE and EXCEPTION mrd's. The program has some lines disabled from the assembler. You can restore these if you wish. The text files are supplied in the TEXT directory. The DEMONSTRATION.S file has a fully documented within itself to help you work out what is happening.

I hope this is useful to you. PLEASE tell me what you think of this software. It does help me to decide whether I should write any more programs such as LOCATE and EXCEPTION mrd's,

kind regards
MICHAEL JOHNSON
11/12 KOKODA ST. WAGGA WAGGA 2650
(069) 255255 (yes I picked it)

ADDRESS ERROR

If all is well you should have an address error printed out, the program has recovered via the restart address, and you can read this information.

You will note that the contents of all the registers have been printed with identifier. The program counter, address being accessed and the instruction opcode are printed on the next line.

The processor function code follows which indicates exactly what the processor was doing when it failed.

The next section gives a fully decoded printout of the status register to simplify the debugging process (this can be very helpful).

The final section of the printout is headed EXTRA INFORMATION. This is the data that you request the system to print as further support in the debugging proccess. All source code of every program on this disc has been put on this disc to assist you in the use of this program by example.

press <ENTER> to continue

dhryreg
dhrynoreg

Description

A benchmark that shows the speed of a compiler and processor combination, especially for relatively small programs. Used for testing speedups in 15 MHz version. `dhryreg` is the fastest version, since it uses register variables where possible. `dhrynoreg` is slower, since it avoids using registers. This is probably Version 1, and has been replaced by a later version on many systems.

The more dhrystones per second, the better.

Examples

Quibbles

Associated files

`dhrystone.c, dhrynoreg.xrel, dhryreg.xrel`

See also

quick.mrd

Distribution

Applix Shareware Disk #9 /15mhz

Author

Reinhold P Weicker
Ported to 1616 by Andrew Morton

dial - simple phone autodialler - Matthew Geier - SW#15

dial phone

Description

A simple telephone auto dialler, that assumes a 2400 baud Hayes compatible modem on `sb:`.

Use Space to retry a number, any other key to abort.

Associated files

`dial.xrel`, `dial.c`,

Distribution

Applix 1616 Shareware Disk # 15 /geier

Author

Matthew Geier. 20/8/1989

disassemble - disassemble memory or file - Gerhard Baumann - SW#14

disassemble `file.xrel` [char]
disassemble `m address1 address2` [char]

Description

`disassemble` will accept either a file or a memory range. The optional `char` parameter will be included in the display of the instruction relative program counter position in the first column, and the opcode in the second column.

The **`m`** informs the program it is to disassemble memory, between address1 and address2.

Associated files

`disassembler.xrel`, `disassembler.s`, `readme`

Distribution

Applix 1616 Shareware Disk # 14 /debug

Author

Gerhard Baumann

diskio - disk input output calls tutorial - Sid Young - SW#6

diskio

Description

Tutorial on using the disk i/o syscalls from assembler.

Quibbles

Associated files

`diskio.exec, diskio.doc, diskio.s`

See also

clrwin, **prog?**

Distribution

Applix 1616 Shareware Disk #6 /tutorial

Author

Sid Young

du2 - select and view disk blocks - Sid Young - SW#6

du2

Description

Select and view the contents of disk blocks. Menu driven, shows filesize, filenames, contents of blocks in ASCII or hexadecimal. Directories can be full or short. Change block numbers, and increment or decrement block number. Logs new disks.

- [a] Ascii block dump.
- [b] Block number decrement.
- [d] Directory.
- [e] Enter block number.
- [h] Hex block dump.
- [i] Information about disk.
- [l] Log a new disk.
- [n] Next block number.
- [s] Short directory.
- [x] E**X**it.

Quibbles

Bombs system badly on blocks 0 or 1.
Has problems with /f1 (Sid doesn't have a second drive).

Associated files

`du2.exec`, `du2.s`, `du2.doc`

See also

rawread, **rawwrite**

Distribution

Applix 1616 Shareware Disk # 6/utilities/sid

Author

Sid Young

easy_write

Description

Michael writes "`Easy_Write` was written to assist in the writing, compiling and testing of source files. It is menu driven to make it easy to use. The program uses a data file `easy_write.data` to store the relevent information for the program being written. `Easy_Write` will compile and/or print files to the printer in the background if you have 1616/OS version 4.0 or later, otherwise these functions will be done in the foreground only. The "compile execute" and "compiler error" files must be set before `Easy_Write` will compile or assemble your source code.

Starting Easy_Write

There are no parameters neccessary, as all the information that `easy_write` requires for your application is stored in the file `easy_write.data` in the current directory. Just type :-

`easy_write` and it will do the rest.

When `easy_write` is started it will present you with the following menu **if** it finds your data file in the **current** directory. If the data file is not found `easy_write` will inform you it was not found before displaying this menu.

```
    MAIN MENU

    EASY WRITE MAIN MENU
    0 EXIT TO 1616 O/S
    1 EDIT A FILE
    2 COMPILE THE FILE(S)
    3 CHECK COMPILER ERRORS
    4 EXECUTE THE PROGRAM
    5 PRINT A FILE TO SCREEN
    6 PRINT A FILE ON THE PRINTER
    7 CHANGE THE DATA FILE
    which (0-7)
```

`0 EXIT TO 1616 O/S`

Will exit to 1616 O/S and release all allocated memory.

`1 EDIT A FILE`

Will print the "EDIT A FILE" heading and give you a menu of all the files which are used by `easy_write` in the current application. To edit the file, simply type the number next to the file, press Enter, and the editor will invoked with that file as the argument. If you wish to set your own tab size, then include the value after the filename, and don’t forget to separate them with a space.

`2 COMPILE THE FILE(S)`

Will execute the string command which is used to compile/assemble the source. The basic design concept of this function is to use a SHELL file to compile/assemble your program. This software is supplied with the source files for `hexagon`. `Cchexagon.Shell` is an example of the way I create the `shell` file that is used to compile `hexagon`. You could use a single command, i.e. `cc easy_write.c }easy_write.err`, instead of the shell file if you wish. However, I recommend the use of a `shell` file for most compilations and assemblies. If you are using version 4.0 or higher, then this function is carried out as a background function. Therefore you must have the error collection file set.

`3 CHECK COMPILER ERRORS`

Will print the contents of the error file onto the screen only. It uses the inbuilt `type` command to do this.

`4 EXECUTE THE PROGRAM`

Will execute the program that you are creating, so that you may test the program during development. The execute filename will be printed, and then `easy_write` waits for you to add any parameters you may require. You then press Enter and the program is started.

`5 PRINT A FILE TO SCREEN`

Will use the inbuilt `type` command to send a copy of the targeted file to the screen. The files will be displayed and selected in the same manner as the `edit` function.

`6 PRINT A FILE ON THE PRINTER`

Will use the inbuilt `type` command to send a copy of the targeted file to the printer. The files will be displayed and selected in the same manner as the `edit` function. If you are using version 4.0 or higher then this function is carried out as a background function.

`7 CHANGE THE DATA FILE`

Will allow you to alter the file `easy_write.data` to reflect any changes that occur e.g. a new source file. See the next section for details.

`CHANGING THE DATA FILE "EASY_WRITE.DATA"`

Function "7" in the main menu will select this function. You are able to set the compile/assemble command (or command file), the execution command and the names of the source files. All source file names must be in full.

```
THE PREPARATION MENU
EASY WRITE CHANGE THE DATA FILE
0 MAIN MENU
1 INSERT a filename for editing
2 DELETE a filename from editing
3 CHANGE COMPILE/ASSEMBLE shell filename
4 CHANGE the EXECUTABLE filename
5 CHANGE the ERROR filename
6 CHANGE a filename for editing
which (0-6)?
```

`0 MAIN MENU`

Will return you to from whence you came (MAIN MENU).

`1 INSERT a filename for editing`

Will prompt you for the filename to be inserted and append the name to the list. If a null string is input then no name is inserted.

`2 DELETE a filename from editing`

Will display the files and wait for you to indicate which one is to be deleted. If a null string is input then no name is deleted.

`3 CHANGE COMPILE/ASSEMBLE shell filename`

Will display the current setting for the compile/assemble file and them prompt for the new file e.g.

Current shell file is 'CCEASY_WRITE'

New shell filename is :-

If a null string is input then no change is made.

`4 CHANGE the EXECUTABLE filename`

Will display the current execution command and then prompt for the new execution command e.g.

Current shell file is 'EASY_WRITE'

New shell filename is :-

If a null string is input then no change is made.

`5 CHANGE the ERROR filename`

Will display the current error file name and then prompt for the new error file name e.g.

Current error file is :- 'EASY_WRITE.ERR'

New error filename is :-

If a null string is input then no change is made.

`6 CHANGE a filename for editing`

Will display the files and them prompt you for the filename to change. This has been included because I make a lot of typing errors. So if you are like me you will need it too. If a null string is input then no name is changed.

Final Comment

This program is supplied ready to use with the `easy_write` program so if you want to try it out and also examine how it works just keep looking. I hope it will all eventually make sense.

Quibbles

Michael will put me out of work, by writing his own superior documentation. This manual page is a direct reprint of his own `doc` file. Great stuff.

Associated files

`easy_write.xrel, easy_write.c, easy_write.data, easy_write.doc`

See also

doc-write

Distribution

Applix 1616 Shareware Disk # 15 /Johnson

Author

Michael Johnson, 11/12 Kokoda St. Wagga Wagga. 2650 (069) 255255

ep232

Description

Menu driven program intended to run a Diamond Systems EP232 eprom burner, via the serial ports. It can handle 2716 to 27512.

a Enter filename

b Bytes, number to read or write

e File offset, for splitting eproms

s Select serial port **sa:** or **sb:**

c compare file to eprom

d exec a 1616/OS command

f hex test eprom

r read eprom into a file

w write eprom from a file

t test the programmer

Set the serial line selects prior to inserting the eprom. The burner is controlled via the RS232 DTR and Rx lines. DTR high means to program an eprom. A low to high transition on DTR resets the address counter.

Quibbles

Associated files

`ep232.exec, ep232.c, ep232.doc, ep232fk.shell` (function key assignments)

Distribution

Applix 1616 Shareware Disk #3 /eprom

Author

Joseph L Moschini, 33 Wade St, Embleton, Perth WA 6062

except - exception handler for bug tracing - Michael Johnson - SW#3, SW#5

except

Description

except is an MRD, and must be installed as part of an MRDriver file at boot up. It is activated by typing **except**, and traces the execution of a program. The source of this program is sometimes called **exception**.

When you write your assembly code, you add **trap#0** (for hard copy) or **trap#1** (for video display) throughout the troublesome areas of your code. **except** handles address errors, bus errors, illegal instructions, and provides a dump of the CPU registers, stack details, a decoded status register, plus user supplied details. These are provided whenever it encounters an appropriate **trap** instruction in your code.

quit continues the trace after you investigate the results of a trap. This is actually the normal 1616/OS command, which continues your program as if it had never stopped.

The document file contains (dare I say it) exceptionally good instructions on the use of the program, and is reprinted in this version of the shareware manual.

Associated files

`except.mrd, except.s, exception.doc, demonstration.s`

See also

trace

Distribution

Applix 1616 Shareware Disk #3 /except

Author

Michael Johnson, 11/12 Kokoda St. Wagga Wagga NSW 2650 (069) 255255 (home,) (069) 230388 (work - a free call).
Suggested donation $10.

Except MRD, by Michael Johnson - SW#3, SW#5

program name:- EXCEPT.MRD
author:- MICHAEL JOHNSON WAGGA WAGGA program source code:- EXCEPT.S
date:- 13-9-88
version :- 2.0
document issue :- 2

EXCEPTION HANDLER

This MRD is used to help find those elusive little bugs which creep into your software. It is designed as a progammers tool, which will supply information concerning your program, its variables, and other information you may wish to know about the software you are developing.

This version is designed as a memory resident driver. Therefore you must build it into the MRDRIVERS file on your disc so that it will be loaded on power up. The MRD will printout a full list of the processors registers and their contents. The status register is printed out in a fully decoded form. A very useful facility is the ability to printout up to 256 long words as EXTRA INFORMATION at the end of the exception printout. You may set this to be any information you wish e.g. program start address (xrel), the value of variable, etc.

The MRD will also accept a restart address which it will use as the return address when the MRD has completed it's functions. This will allow you to install a crash recovery system into your software if needed.

A full source listing is supplied so you can modify it if you wish. The exception handler is memory independent, therefore if you do modify it please make sure you maintain the memory independence. The current version of SSASM will produce an XREL format file on one assembly so you do not need to use the genreloc.xrel package to produce an MRD file. DO NOT alter the first few dc.l statements or the last dc.l statement and DON'T put any code after the last dc.l statement (progend) or you will not produce an XREL file.

This MRD will use a stack frame of 36K bytes so make sure you allow enough space for the stack when you install this driver.

EXCEPTIONS HANDLED BY EXCEPTION MRD

The following exceptions are handled by the EXCEPTION MRD

(1) Address Error
(2) Buss Error
(3) Illegal Instructions
(4) Trap Zero (for investigating bugs with hardcopy)
(5) Trap One (for investigating bugs with softcopy)

The first 4 exceptions will give you a hard copy of the proccessor register contents, the information placed on the stack by the exception and any extra information you may want in order to find the problem. The first 3 also provide a facility whereby your software may be restarted from a given point. You simply supply a restart address for this facility to operate, otherwise it will execute a warm boot of 1616 O/S. The trap zero and trap one exceptions will continue execution of your software with the entire processor registers and status restored to the same condition they were in when the instruction was reached by the program counter.

INFORMATION SUPPLIED

The exception handler is designed to give you as much information as possible. The following information is printed:-

1) all CPU register values.
2) The additional information supplied on the stack for a buss or an address error
3) the function code fully decoded
4) the status register fully decoded
5) any extra information (user supplied)

When the printout has been completed the following will occur.

If a buss error, address error or illegal instruction had been encountered the following happens:-

1) adjust the return address on the stack (explained later)
2) return via the address it placed on the stack in the previous step

If a trap one or trap zero was encountered the status register and program counter are reloaded from the stack and your program will then continue as if it had never stopped.

ADDRESS ERROR

When an address error occurs the contents of all the registers will be printed with with their identifiers. The program counter, address being accessed and the instruction opcode are printed out. The processor function code follows which indicates exactly what the processor was doing when it failed. The next section gives a fully decoded printout of the status register to simplify the debugging process (this can be very helpful). The final section of the printout is headed EXTRA INFORMATION. This is the data that you request the system to print as further support in the debugging proccess.

example of the address error printout :-

```
ADDRESS ERROR
    D0=12345678   D1=0000000C   D2=00000002   D3=FFFFFFFF
    D4=0003B102   D5=0000000D   D6=000003E8   D7=00000003
```

```
    A0=0003AFD8   A1=0003ACF8   A2=0003AD0C   A3=0003E538
    A4=0003E545   A5=0003AFD8   A6=000540BC   A7=0005405E

PROGRAM COUNTER:- 0003AB14     ADDRESS:- 0003AFD9
INSTRUCTION:- 2028

FUNCTION:-read supervisor data

STATUS:-
trace bit=0     supervisor bit=1   interrupt level=0
extend bit=0
negative bit=0            zero=0      overflow bit=0
carry bit=0

EXTRA INFORMATION
     31313131    32323232      33333333

END OF PRINT
```

BUSS ERROR

This exception will produce the same printout as the address error and all other functions are the same.

ILLEGAL INSTRUCTION ERROR

This exception will produce a printout whenever the processor tries to execute an opcode which is not valid (except the unimplemented instructions).

The printout is as follows:-

```
ILLEGAL INSTRUCTION
    D0=12345678   D1=0000000C   D2=00000002   D3=FFFFFFFF
    D4=0003B102   D5=0000000D   D6=000003E8   D7=00000003

    A0=0003AFD8   A1=0003ACF8   A2=0003AD0C   A3=0003E538
    A4=0003E545   A5=0003AFD8   A6=000540BC   A7=0005405E

PROGRAM COUNTER:- 0003AB14

STATUS:-
trace bit=0    supervisor bit=1    interrupt level=0
extend bit=0
negative bit=0           zero=0       overflow bit=0
carry bit=0

EXTRA INFORMATION
     31313131    32323232      33333333

END OF PRINT
```

TRAP ZERO (hard copy)

This instruction will print the data on the printer and then return to the next instruction following the trap zero instruction which called it with all of the proccessor registers and status register restored exactly as they were before the trap zero was executed.

In other words:-

THE PROGRAM WILL EXECUTE TRAP ZERO THEN CONTINUE EXECUTION AS IF IT HAD NEVER HAPPENED!

an example of the printout:-

```
TRAP ZERO
   D0=12345678   D1=0000000C   D2=00000002   D3=FFFFFFFF
   D4=0003B102   D5=0000000D   D6=000003E8   D7=00000003

   A0=0003AFD8   A1=0003ACF8   A2=0003AD0C   A3=0003E538
   A4=0003E545   A5=0003AFD8   A6=000540BC   A7=0005405E

PROGRAM COUNTER:- 0003AB14

STATUS:-
trace bit=0  supervisor bit=1      interrupt level=0
extend bit=0
negative bit=0         zero=0         overflow bit=0
carry bit=0

EXTRA INFORMATION
     31313131    32323232      33333333

END OF PRINT
```

TRAP ONE (soft copy)

This exception is different to the others. It will print all the information on the SCREEN. The current screen contents are saved on the stack (all 32K). The current window, video display page and cursor information is saved. Then the information is printed. Trap one will NOT print any extra information. When the data is printed the handler will then drop into the 1616 O/S command handler so that you may examine memory or whatever.

When you are satisfied you simply type QUIT and the entire screen is reloaded from the stack and restored to its original state. The processor is then reloaded to its condition prior to the exception and your software continues as if the exception had never happened. The following steps take place when a trap one is executed:-

1) save the video display page
2) save the screen mode
3) save screen window
4) make sure access page = display page
5) save the screen on the stack (all 32K)
6) set mode to 640
7) set new window
8) clear the screen
9) find the screen driver
10) setup the 4K print buffer on the stack
11) create the exception string in the buffer
12) print the string
13) set new prompt
14) drop into 1616o/s

At this point you may execute any 1616o/s commands, and then the user types QUIT (hopefully), and the story continues:-

15) unlink the print buffer from the stack
16) restore users window
18) restore users screen mode
19) restore the video display page
20) exit the exception

At this point we should be back inside the users program, with the entire system in exactly the same state it was just before the trap #1 was executed.

RESTART POINTER

You will need to set this to the restart address location within your program if you wish to include a crash recovery section in your software. If an exception other than the trap zero or one occurs then this address is used to replace the program counter address on the stack. The RTE instruction then loads this address as the return address.

The MRD will set up a pointer to its own internal data structure which will set this address to an internal routine which executes a warm boot of the system (level 2 reset). If you wish to supply extra information but not the restart address then set the restart address to zero and the MRD will do a warm boot after printing the extra information.

The EXTRA INFORMATION

The handler will allow you to set up an address table which point to long words which will be printed. The maximum number of long words you can set is 256. All output is directed to the centronics port except for the trap one exception which will print all information, except the user supplied extra information, on the screen.

You will be required to supply a pointer to a structure within your program in order to use this facility. If this pointer is not supplied then the MRD will set the pointer to an internal structure which will do a warm boot of the system. The data structure is as follows:-

long word
1) the restart address for your code
2) the number of extra words to be printed
3) address of first long word to be printed
4) address of second long word to be printed

etc.....

N) address of last long word to be printed

HOW TO USE THE EXCEPTION MRD

THE MRD COMMAND FUNCTIONS AND USES

RESETS

A level 0 reset steps are:-

1) The MRD will read in the current 1616o/s settings and save them.
2) setup a table with the MRD vector values in it.
3) set state to enabled
4) set the data structure pointer to internal data structure.

The level 1 and 2 resets will

1) set the state to active
2) set the data structure pointer to internal data structure.

ENABLE

The enable command will enable the mrd if it has been disabled. No argument is required. The state is set to active. Returns 0 on exit

DISABLE

The disable command will prevent the mrd from being used by the 1616o/s level call or the callmrd() syscall. The state is set to passive. Returns 0 on exit.

DOIT

The doit command requires an argument which is a pointer to your data structure or set to zero if you do not wish to set your own data structure. The MRD will then return a zero if your data structure was used or a one if the MRD's internal data structure was used or -1 if the mrd is inactive. The following steps occur.

1) the argument is tested for a valid address i.e. $4000 < pointer value < stack pointer
2) set the restart pointer value (argument from user if valid)
   set internal data structure if argument = 0 or invalid.
3) copy the MRD vector pointers to the MC68000 vector table
4) return a value of 0, 1 or -1.

STOPIT

The MRD's activity is stopped when this command is sent to it. No argument is required. The following steps are taken when this command is received. This call will return 0 if mrd is active or -1 if the mrd is not active.

1) set pointer to internal data structure
2) copy the 1616o/s vector pointers to the MC68000 vector table
3) return 0 or -1.

VERSION

This command will return the version of the MRD as documented in 1616 technical reference manual. The first MRD version is 2.0 (hopefully this will not need any updating).

NAME

This command will return a pointer to the MRD's name EXCEPTION as documented in the 1616 technical reference manual.

EXECENTRY

This command will return a pointer to the MRD's name command level entry address as documented in the 1616 technical reference manual.

1616O/S COMMAND LEVEL USAGE

The simplest way is to use the MRD is via the 1616o/s command EXCEPTION. The mrd will then give you a status printout to say which vectors are set. If the MRD's vectors were set then this command will reset the 1616o/s vector values. If the 1616o/s vectors were set then the command will set the MRD's vectors.

USING THE MRD FROM WITHIN YOUR SOFTWARE (assembler)

The first step is to make sure that the EXCEPTION MRD is loaded. To do this we look for it by name and at the same time we also find its call value and store it. All of the following code is written to be relocatable without any changes.

First we must find the MRD. If it is not present then we can't use it. The following equates are used in this example.

```
callmrd             equ         89          call mrd syscall
doit   equ          7           doit mrd command
stopit equ          8           stopit mrd command
mrdname             equ         3           find mrd name
```

* find the EXCEPTION mrd

```
        lea         name(pc),a2             get address of name
        move.l      #0,d7       mrd's are zero numbered
        move.l      #mrdname,d2             get name command
tstmrd  move.l      d7,d1       load the MRD number
        move.l      #mrdname,d2             command to get the
name pointer
        move.l      #callmrd,d0             call the driver
        trap        #system
        tst.l       d0          found one
        bmi         retrn       no must have been through them
all
        * we have a name now test for a match
```

```
       move.l      d0,a0       mrd's name from callmrd
       move.l      a2,a1       name we are looking for
cmptst cmp.b       (a0)+,(a1)+             test character by
character
       bne         next        not this mrd try next
       tst.b       (a0)        end of name
       beq         found       yes we found it
       bra         cmptst      no test next character

next   addq.l      #1,d7       no wrong one try the next
       bra         tstmrd

* name of MRD

name   dc.b        'EXCEPTION',0,0         exception handler
```

This section of code is used when the MRD cannot be found. You will need to formulate this section to fit your situation.

```
retrn  ????        code to handle this situation
```

The MRD has been found so now you setup the data structure and insert the extra information and restart address. You put the address of each word to be printed onto the stack in reverse order. This allows us to write relocatable code (memory independent). NOTE start1 is the address of the instruction at which we continue execution after the exception.

```
found  lea         mrdnum(pc),a0           save MRD number at
this location
       move.l      d7,(a0)     save number
       move.l      #3,d0       print 3 long words (for this
example)
       pea         data3(pc)   third long word
       pea         data2(pc)   second long word
       pea         data1(pc)   first long word
       move.l      d0,-(sp)    number of words to print
       pea         start1(pc)  address of restart point
       lea         restart(pc),a0          pointer to data
structure

* push the information into data structure

       move.l      (sp)+,(a0)+             insert restart
address
loop   move.l      (sp)+,(a0)+             from stack to struc-
ture
       dbra        d0,loop     this will count the no.  of
words
```

The MRD must be activated. The doit command requires an argument which is a pointer to the data structure.

```
       lea         restart(pc),a0          address of data
structure (argument)
       lea         mrdnum(pc),a1           get address of
number
       move.l      (a1),d1     load the MRD number
```

```
        move.l      #doit,d2    command to activate it
        move.l      #callmrd,d0             call the driver
        trap        #system
```

Here you test the returned value for the setting or maybe the MRD is disabled.

```
        tst.l       d0          test return value
        bmi         disabled    mrd is disabled
        beq         mypointer   my pointer was set

* the system pointer was set
```

At this point the MRD has been activated and loaded with the information it requires to operate. If you wish to activate it and not supply the argument then make the argument zero and the MRD will setup its own data structure. This data structure will not give any extra information and will do a warm boot of 1616o/s when the exception has been printed. The trap one and trap zero instructions will return to your program with the proccessor intact and continue execution from the next instruction reguardless of which data structure has been set.

BEFORE YOU EXIT BACK TO 1616O/S

IF you have activated the MRD then you must deactivate the MRD before you exit from your program. This is a very simple procedure which requires no argument. The following code will achieve this :-

* deactivate the EXCEPTION mrd and reset the structure pointer and
* restore 1616o/s vectors in the vector table

```
        lea         mrdnum(pc),a1
        move.l      (a1),d1     mrd number
        move.l      #stopit,d2  stopit will reset the
*                               data structure pointer
        move.l      #callmrd,d0             call the driver
        trap        #system
```

You are able to exit from your program safely at this point.

DATA STRUCTURE FOR EXTRA INFORMATION

The following data structure is used by the EXCEPTION MRD to print the extra information and it also holds the restart address. The first long word is the restart address if this address is zero then the system will do a warm boot after handling the exception (but a trap zero or trap one will simply return to the program which called it). The second long word is the number of extra long words to be printed and if this is equal to zero then none will be printed. The third and subsequent number of long words are the addresses of the extra long words which will be printed.

```
restart             dc.l        0           restart address
        dc.l        0           set for number of extra long
words to print
        dc.l        0           first address
        dc.l        0           second address
        dc.l        0           third address
```

```
*       etc.......  until last long word is defined
        dc.l        0           nth address
```

VERY IMPORTANT

NOTE THIS
It will use 36K OF STACK, so make sure you extend the stack size when you install this MRD.

register usage by the exception handler

a0 general purpose
a1 as output buffer pointer
a2 as pointer to hex table
a4 as pointer to regdat (initial load data)
  holds stack frame info
a5 as pointer to string table
a6 copy of stack pointer/reg pointer
d0 as temp store for long word/general use
d1 as the hex byte accum.
d2 as counter for no. of hex bytes to print
d3 as counter for no. of regs
d4 as store of " d0=" (reg name)
d5 as store for "0c0d" (cr,lf)
d6 as store for " " 2 spaces
d7 general use

This is the internal data structure which is used if no user data has been supplied. A zero restart address will set the address of the internal warm boot code on the stack as the return address.

```
restrt dc.l        0           internal restart
*       do a warm boot (always zero)
        dc.l        0           no extra information to print
*       (always zero)
```

USING THE MRD FROM WITHIN YOUR SOFTWARE (C compiler)

THE DATA STRUCTURE

This the data structure used by C. note that they are all pointers except the actual number_of_variable which is an int.

```
struct data_restart
{
        int         *restart;
        int         number_of_variables;
        int         *var1;
        int         *var2;
        int         *var3;
} exception_data;
```

The first thing we do is find the mrd. If it is not there then we cannot use it.

```
char   *name;
int    mrd_number;

name = "EXCEPTION";

mrd_number = find_mrd(name);
```

This function is used to find an mrd by name if it present it will return the number of the mrd each mrd in turn starting from zero. If it cannot find the mrd then it will return -1 (a zero would be mrd zero).

```
int    findmrd(name)          /* find an MRD by name */
char   *name;      /* points to mrd's name */
{
char   *mrd_name;  /* pointer returned by get mrd name */
int    mrd_numb = 0;          /* count the MRD's as we go */

/* the while expression will test if this number MRD exists */

while( ( (int) ( mrd_name = get_mrd_name(mrd_numb) ) ) > 0)
       {
       if ( strcmp(name, mrd_name) )  /* is this the one */
                     mrd_numb += 1;          /* no try the next
*/
       else
                     return(mrd_numb);       /* yes return its
number */
       }
return(-1);        /* MRD not found so say so */
}
```

The next step is to see if it was found.

```
if (mrd_number != -1)
       {
       printf("\n%s mrd number is %d\n\n",name, mrd_number);
```

At this point we know the mrd is present and we can use it. You must activate the mrd before you can use it.

```
mrd_set = doit_mrd(mrd_number, &exception_data);
```

You insert whatever else you need here. You use the restart() function to set your restart address. Be very carefull as this function will return ZERO if a restart has occurred. If you have have called it then it will return the restart address. You must call it TWICE the first time sets the address in exception-_data structure, the second gives you the RESTART indication.

```
exception_data.restart = restart();
```

When a restart occurrs the exception handler will effectively return to this expression and complete it. So save_restart_address will have ZERO if a restart has occurred OR non zero if no restart has occurred.

```
save_restart_address = restart();         /* exception restart
*/
```

Test for a restart.

```
{
/* setup the mrd to our data */
```

At this point we have been passed a valid pointer for the restart address THEREFORE you are still setting up the MRD.

```
mrd_set = doit_mrd(mrd_number, &exception_data);
```

This enormous function actually activates the exception handler. The argument value is the address of the exception_data structure if you are using the full capabilities of the exception handler. IF not then set the argument to zero and the restart will be a reset (level 2 if you're lucky OR level 0 if you're like me) (once the reset button would not get me back to 1616o/s but that's another story and it's NOT tall).

```
doit_mrd(number, argument)     /* activate the mrd */
int    number, argument;
{
int    command = 7;            /* doit */
return( (int) callmrd(command, number, argument) );  /* do it
*/
}
```

This is where you need to find out which vectors have been set IF this is important to your particular situation. If it is not important and you don't need it then don't do this. The printf() functions would be replaced with some other code supplied by you (SORRY I can't do all the work).

```
if(mrd_set)
printf("system structure is set\n\n");
else
printf("user structure is set\n\n");
}
else
{
```

At this point you know that a restart has occurred so you will have to supply the code to cope with this situation.

```
printf("a restart has occurred\n\n");
}
}
else               /* else from if ( save_restart_address ) */
{
```

At this point the mrd has not been found. It is up to you to survive this situation. (YOUR problem not mine SORRY).

```
}
```

This function is used to set the data for the exception handler to print. This is a poor example of how to do it but it does work and I am sure that you will work out a better way because I have not had the time (sound familiar).

```
set_exception_data(val1, val2, val3)
void   *val1, *val2, *val3;
{
exception_data.number_of_variables = 3;   /* print this many
variables*/
exception_data.var1 = val1;
exception_data.var2 = val2;   /* these are pointers to vari-
ables */
exception_data.var3 = val3;
}
```

When you have finished with the handler or intend to exit from your software then you must do the following.

DON'T FORGET THIS it may save you a lot of late nights.

```
stopit_mrd( mrd_number );     /* finished so we deactivate mrd
*/

/* this function can save a lot of late nights (maybe) */

stopit_mrd( number )          /* if you can't work it out it's
too LATE go to bed */
int    number;
{
int    command = 8;           /* stopit */

callmrd( command, number, 0); /* no argument for stopit */
}
```

The source code TEST.C demonstrates the use of this mrd. It was actually used to test the mrd and serves as a good example of how to use the exception handler and the locate mrd's. You may notice that this description is a direct extract from that code.

PUTTING IT ALL TOGETHER

```
main()
{
char   *name;
static int         *save_restart_address;
static int         mrd_number;
static int         mrd_set;

       name = "EXCEPTION";
       set_exception_data(&exception_data, &mrd_number,
&name);
       exception_data.number_of_variables = 3;
       mrd_number = findmrd( name );

       if (mrd_number != -1)
                  {
                  printf("\n%s mrd number is %d\n\n",name,
mrd_number);
                  exception_data.restart = restart();
                  save_restart_address = restart();
```

```
                    if ( save_restart_address )
                                {
                                mrd_set = doit_mrd(mrd_number,
&exception_data);
                                if(mrd_set)
                                            printf("system
structure is set\n\n");
                                else
                                            printf("user struc-
ture is set\n\n");
                                }
                    else
                                {
                                printf("a restart has occur-
red\n\n");
                                stopit_mrd(mrd_number);
                                exit(0);
                                }
                    }
        else
                    printf("the %s mrd not found\n", name);
        stopit_mrd( mrd_number );
}

int     findmrd(name)
char    *name;
{
char    *mrd_name;
int     mrd_numb = 0;
        while( ( (int) ( mrd_name = get_mrd_name(mrd_numb) ) )
> 0)
                    {
                    if ( strcmp(name, mrd_name) )
                                mrd_numb += 1;
                    else
                                return(mrd_numb);
                    }
        return(-1);
}
char    *get_mrd_name(number)
int     number;
{
int     mrd_name = 3;
        return( (char *) callmrd( mrd_name, number, 0 ) );
}
void    *callmrd(command, number, argument)
int     command, number, argument;
{
int     callmrd = 89;
        return( (char *) trap7(callmrd, number, command, argu-
ment));
}
```

```
stopit_mrd( number )
int    number;
{
int    command = 8;

       callmrd( command, number, 0);
}

doit_mrd(number, argument)
int    number, argument;
{
int    command = 7;

       return( (int) callmrd(command, number, argument) );
}

set_exception_data(val1, val2, val3)
void   *val1, *val2, *val3;
{
       exception_data.number_of_variables = 3;
       exception_data.var1 = val1;
       exception_data.var2 = val2;
       exception_data.var3 = val3;
}
```

I hope you find this program useful.

factorial - evaluates factorials - Gerhard Baumann - SW#15

factorial [-v] number

Description

Evaluates factorials, provided they do not exceed 32768. On a 15 MHz 1616, factorial 100 took 1.82 seconds, 200 took 7.52 seconds, 300 took 17.66 seconds. I advise you leave large numbers to run overnight, and redirect output to disk.

-v verbose output.

Associated files

factorial.xrel, factorial.s,

Distribution

Applix 1616 Shareware Disk # 15 /baumann

Author

Converted from BASIC version in *Your Computer*, March 1989 by Gerhard Baumann

ftree - find files and display directory trees - Michael Johnson - SW#15

ftree [-s filename][-d][-f][/dev]

Description

-s `filename` will search the device(s) specified for the file/sub-directory `filename` and print all paths where it is found. The '*' can be used in a filename as a wildcard. The name must be enclosed in quotes if you use the wildcard. This will prevent the system from expanding the wildcard within the current directory. NOTE: you must insert a space between `-s` and the filename

-d print path and sub-directories only, no file names

-d print path and files only, no sub-directory names will be printed

/devuse this device(s), OR alter device table(s) OR use this directory only (relative/absolute)

USE The -s -d -f are mutually exclusive. If one is specified then the rest are not. e.g.

`ftree -s ftree.xrel` will only search for the filename `ftree.xrel`.

`ftree -f` will only print the paths and filenames.

`ftree -d` will only print the paths and directories.

`ftree -s ftree.xrel -d` will print the paths and directories AND search for `ftree.xrel`.

`ftree` will print all sub-directories and filenames.

`ftree /h1/shell` will print all sub-directories and filenames starting at sub-directory `shell` on device `/h1`.

NOTE: If there are no filenames/sub-directories to print then the pathname will not be printed.

`/dev` specifies which device(s) to search OR to change the device table(s)

PARAMETERS

`filename` any valid filename

`/dev` any valid device name, OR

`/all` to search all devices

`/fx` to search all floppy disc devices

`/hx` to search all hard disc devices

`/cdt` to change the device tables

NOTE: does not support volume names.

If device = `/cdt` then a menu is displayed so that you may set the devices file to suit your system. If you do not have a hard disc then set the hard disc with a null input. The new devices will be used if you have specified `/all /fx` or `/hx` on the command line.

DEVICES FILE

The devices are saved under the filename `/SYSDEV/FTREE.DEV` you will have to assign `/SYSDEV` to a directory within your system. I suggest you insert that assignment in your `AUTOEXEC.SHELL` file. If you do not have a hard disc then I suggest you boot with a disc which will create this directory in your RAM disc and assign `/SYSDEV` to this directory.

Examples

`ftree` will search the current device and print all pathnames, sub-directories and files on that device.

`ftree -d` will search the current device and print all pathnames and sub-directories on that device.

`ftree -f` will search the current device and print all pathnames and filenames on that device.

`ftree -d /f0` will search floppy zero and print all pathnames and sub-directories on floppy zero.

`ftree -s ftree.doc` will search the current device for the file `ftree.doc` and print all pathnames in which it is found. IF it is a sub-directory then the pathname will be printed and the file will be shown as a sub-directory.

`ftree -s ftree.doc /f0/special/doc` will search the current device for the file `ftree.doc` and print all pathnames in which it is found. IF it is a sub-directory then the pathname will be printed and the file will be shown as a sub-directory. It will start at sub-directory `/f0/special/doc` and search down from there.

`ftree -s ftree.doc /fx` will search all floppies for the file `ftree.doc` and print all pathnames in which it is found.

```
ftree /all will search all devices in the device file and
print all pathnames sub-directories and files on ALL devices.
CAUTION: will create heaps of output if you have a hard disc.
```

`ftree /.` will search from the current directory and print all pathnames sub-directories and files.

`ftree /..` will search from the previous directory and print all pathnames sub-directories and files.

`ftree /../work` will search from sub_directory `WORK` in the previous directory and print all pathnames sub-directories and files.

```
ftree /cdt -s ftree.doc /all
ftree -s ftree.doc /all /cdt
ftree /cdt /all -s ftree.doc
ftree /all /cdt -s ftree.doc
```
The four commands above will all do exactly the same thing.
1) open the "devices file" and allow you to edit the file.
2) search the devices in the altered (created) devices file for the file `ftree.doc` and report all pathnames in which it was found.

`ftree -d -s ftree.doc`
1) searches the current device for the file `ftree.doc` and prints all pathnames under which it is found.
2) prints all pathnames and sub-directories on the current device. NOTE: These two functions are done concurrently and not consecutively.

`ftree /cdt` will allow you to alter or create the device table in the devices file. No output information will be printed from any devices.
1) floppy devices
2) hard disc devices
3) all devices

`ftree -s "ftr*"` will search the current device for all sub-directories and file-names that start with "FTR" and print their pathname. e.g. `FTREE.C FTREE.DEV FTREE.XREL`

`ftree -s "*ee*.c` will search the current device for all sub-directories and filenames that include "ee" and print their pathname. e.g. `FTREE.C SCREEN.C`

Associated files

`ftree.c, devices.c, top-print.c, ftree.doc, /sys-dev/ftree.dev`

Distribution

Applix 1616 Shareware Disk #15 /johnson

Author

This version 2.2, 31/7/1989, suggested donation $10. Michael Johnson, 11/12 Kokoda St, Wagga Wagga, NSW 2650, (069) 255255

getty - restricted shell via serial port - Andrew McNamara - SW#15

getty [sb:] [-u*id*] [-m*alrwx*] [-c*barfile*]

Description

Makes it easy to run multiple users on a 1616, by providing an (optionally) restricted shell via one of the serial ports. Use `getty` to start ***iexec*** on serial port `sa:` (default) or (optionally) on `sb:`.

-u *ID* Select a User Identification number for the new shell. 0 is normally unrestricted, 1 to 65535 can be used.

-m *Umask* to be set for the new shell. The options here are the normal directory attribute bits for files, and determine the default attribute condition of new files.

- **a** Archive bit set.
- **l** Locked file bit set.
- **r** Read by owner only.
- **w** Write by owner only.
- **x** EXecute by owner only.

-c `barfile` is simply a file containing the list of 1616 commands to be locked out of use for this particular shell. See `fred` for an typical lockout file.

Examples

`getty`
Use serial port sa:, with all commands allowed.
`getty sb:`
Use serial port sb:, with all commands allowed.
`getty sa: -cfred`
Use serial port sa: The file `fred` contains a list of all commands to lock out of action.

Quibbles

Associated files

`getty.c, getty.use, getty.xrel, fred`

Distribution

Applix 1616 Shareware Disk #15 /McNamara

Author

Andrew McNamara

hs - Help for system calls - Conal Walsh - SW#13

hs `name` *or* `number`

Description

A quick and dirty program to aid you in recalling how the various 1616/OS system calls are used.

The `name` can be any 1616/OS call name (or an abbreviation of one), while `number` can be any 1616/OS number.

Examples

`hs freet`	gives help for ***freetone*** call.
`hs .31`	gives help for ***set_640*** call.

Quibbles

You will probably have to `chmem` it prior to use.
Wom't work on early version 1616/OS.

Associated files

`-readme, c1616.c, c1616.xrel, convhc.c, convhc.xrel, h.bak, h.shell, hcgrap.csh, hcgrap.rch, help.bak`, all in /c1616, which is a later highly developed generalised version.

`-readme, helpsys.as, hs.xrel`

See also

c1616

Distribution

Applix 1616 Shareware Disk #13 /helpsys

Author

Conal Walsh

indent `[input-file [output-file]] [-bad | -nbad] [-bap | -nbap] [-bbb | -nbbb] [-bc | -nbc] [-bl] [-br] [-cn] [-cdn] [-cdb | -ncdb] [-ce | -nce] [-cin] [-clin] [-dn] [-din] [-fc1 | -nfc1] [-in] [-ip | -nip] [-ln] [-lcn] [-lp | -nlp] [-pcs | -npcs] [-npro] [-psl | -npsl] [-sc | -nsc] [-sob | -nsob] [-st] [-troff] [-v | -nv]`

Description

Indents and formats C program source. It reformats the C program in the `inputfile` according to the numerous switches. If no output filename is given, a backup copy of the input file is made with the extension `.bak`, and the input file name is used for the output file. If `outputfile` is specified, `indent` checks that the name is different to `inputfile`.

The numerous switches can appear before or after the file names.

NOTE: If you only specify an input-file, the formatting is done ‘in-place’, that is, the formatted file is written back into input-file and a backup copy of input-file is written in the current directory. If input-file is named `/blah/blah/file`, the backup file is named `file.BAK`.

If output-file is specified, indent checks to make sure it is different from input-file.

OPTIONS The options listed below control the formatting style imposed by indent.

-bap,-nbap — If -bap is specified, a blank line is forced after every procedure body. Default: -nbap.

-bad,-nbad — If -bad is specified, a blank line is forced after every block of declarations. Default: -nbad.

-bbb,-nbbb — If -bbb is specified, a blank line is forced before every block comment. Default: -nbbb.

-bc,-nbc — If -bc is specified, then a newline is forced after each comma in a declaration. -nbc turns off this option. The default is -bc.

-br,-bl — Specifying -bl lines up compound statements like this:

```
if (...)
{
    code
}
```

Specifying -br (the default) makes them look like this:

```
if (...) {
    code
}
```

-cn — The column in which comments on code start. The default is 33.

-cdn — The column in which comments on declarations start. The default is for these comments to start in the same column as those on code.

-cdb,-ncdb — Enables (disables) the placement of comment delimiters on blank lines. With this option enabled, comments look like this:
/*
* this is a comment
*/
Rather than like this:
/* this is a comment */
This only affects block comments, not comments to the right of code. The default is -cdb .

-ce,-nce — Enables (disables) forcing ‘else’s to cuddle up to the immediatly preceeding ‘}’. The default is -ce .

-cin — Sets the continuation indent to be n. Continuation lines will be indented that far from the beginning of the first line of the statement. Parenthesized expressions have extra indentation added to indicate the nesting, unless -lp is in effect. -ci defaults to the same value as -i.

-clin — Causes case labels to be indented n tab stops to the right of the containing switch statement. -cli0.5 causes case labels to be indented half a tab stop. The default is -cli0.

-dn — Controls the placement of comments which are not to the right of code. The default -d1 means that such comments are placed one indentation level to the left of code. Specifying -d0 lines up these comments with the code. See the section on comment indentation below.

-din — Specifies the indentation, in character positions, from a declaration keyword to the following identifier. The default is -di16 .

-fc1,-nfc1 — Enables (disables) the formatting of comments that start in column 1. Often, comments whose leading ‘/’ is in column 1 have been carefully hand formatted by the programmer. In such cases, -nfc1 should be used. The default is -fc1.

-in — The number of spaces for one indentation level. The default is 4.

-ip,-nip
: Enables (disables) the indentation of parameter declarations from the left margin. The default is -ip .

-ln
: Maximum length of an output line. The default is 75.

-npro
: Causes the profile files, ‘./.indent.pro’ and ‘~/.indent.pro’, to be ignored.

-lp,-nlp
: Lines up code surrounded by parenthesis in continuation lines. If a line has a left paren which is not closed on that line, then continuation lines will be lined up to start at the character position just after the left paren. For example, here is how a piece of continued code looks with -nlp in effect:

```
    p1 = first_procedure(second_procedure(p2,
p3),
            third_procedure(p4, p5));
```

With -lp in effect (the default) the code looks somewhat clearer:

```
    p1 = first_procedure(second_procedure(p2,
p3),
                                third_procedure(p4,
p5));
```

Inserting a couple more newlines we get:

```
p1 = first_procedure(second_procedure(p2,
                                                p3),
                                    third_procedure(p4,
                                                   p5));
```

-pcs , -npcs
: If true (-pcs) all procedure calls will have a space inserted between the name and the ’(’. The default is -npcs

-psl , -npsl
: If true (-psl) the names of procedures being defined are placed in column 1 - their types, if any, will be left on the previous lines. The default is -psl

-sc,-nsc
: Enables (disables) the placement of asterisks (‘*’s) at the left edge of all comments.

-sob,-nsob
: If -sob is specified, indent will swallow optional blank lines. You can use this to get rid of blank lines after declarations. Default: -nsob

-st
: Causes indent to take its input from stdin, and put its output to stdout.

-Ttypename
: Adds `typename` to the list of type keywords. Names accumulate: -T can be specified more than once. You need to specify all the typenames that appear in your program that are defined by typedefs - nothing will be harmed if you miss a few, but the program won’t be formatted as nicely as it should. This sounds like a painful thing to have to do, but

it's really a symptom of a problem in C: typedef causes a syntactic change in the laguage and indent can't find all typedefs.

-troff — Causes indent to format the program for processing by troff. It will produce a fancy listing in much the same spirit as vgrind. If the output file is not specified, the default is standard output, rather than formatting in place.

-v,-nv — -v turns on 'verbose' mode, -nv turns it off. When in verbose mode, indent reports when it splits one line of input into two or more lines of output, and gives some size statistics at completion. The default is -nv.

FURTHER DESCRIPTION

You may set up your own 'profile' of defaults to indent by creating a file called .indent.pro in either your login directory or the current directory and including whatever switches you like. A '.indent.pro' in the current directory takes precedence over the one in your login directory. If indent is run and a profile file exists, then it is read to set up the program's defaults. Switches on the command line, though, always override profile switches. The switches should be separated by spaces, tabs or newlines.

Comments

'Box' comments. Indent assumes that any comment with a dash or star immediately after the start of comment (that is, '/*-' or '/**') is a comment surrounded by a box of stars. Each line of such a comment is left unchanged, except that its indentation may be adjusted to account for the change in indentation of the first line of the comment.

Straight text. All other comments are treated as straight text. Indent fits as many words (separated by blanks, tabs, or newlines) on a line as possible. Blank lines break paragraphs.

Comment indentation

If a comment is on a line with code it is started in the 'comment column', which is set by the -cn command line parameter. Otherwise, the comment is started at n indentation levels less than where code is currently being placed, where n is specified by the -dn command line parameter. If the code on a line extends past the comment column, the comment starts further to the right, and the right margin may be automatically extended in extreme cases.

Preprocessor lines

In general, indent leaves preprocessor lines alone. The only reformmatting that it will do is to straighten up trailing comments. It leaves imbedded comments alone. Conditional compilation (#ifdef...#endif) is recognized and indent attempts to correctly compensate for the syntactic peculiarites introduced.

C syntax

Indent understands a substantial amount about the syntax of C, but it has a ‘forgiving’ parser. It attempts to cope with the usual sorts of incomplete and misformed syntax. In particular, the use of macros like: #define forever for(;;) is handled properly.

BUGS

Indent has even more switches than ls.

A common mistake that often causes grief is typing:indent *.c to the shell in an attempt to indent all the C programs in a directory. This is probably a bug, not a feature.

Associated files

```
args.c, indent.1, indent.c, indent.man, indent.shell,
indent_codes.h, indent_globs.h, io.c, lexi.c, makefile,
parse.c, pr_comment.c, -indent.xrel
```

Distribution

Applix 1616 Shareware Disk # 10 /indent 192 blocks

Author

Ported to 1616 by Andrew Morton.

kmem - C storage allocator - Andrew Morton - SW#6

kmem

Description

C storage allocator for Z80 and other 8 bit systems.

Associated files

`kmem.c`,

Distribution

Applix 1616 Shareware Disk # 6

Author

Andrew Morton.

kv - prints keyboard scan codes - Andrew Morton - SW#6

kv

Description

Prints the keyboard scan codes produced by your particular keyboard.Given the somewhat strange results some "standard" keyboard produce, this could be handy when you are writing something that needs to know what the keyboard really produces.

Quibbles

Associated files

`kv.exec, kv.s`

See also

scan Jeremy Fitzhardinge also produced a keyboard scan code program.

Distribution

Applix 1616 Shareware Disk #6 /tutorial

Author

Andrew Morton(?)

lfk - define function keys - Cameron Hutchison - SW#15

lfk `fkey_def_file`

Description

Takes a file with function key definitions in it, and programs the function keys specified.

If the last character in a line is | replaces it with a carriage return.

The `fkey_def_file` contains lines as follows:
*n*F_key_def
where *n* is from 1 to 0 (0 is F10). There must be a carriage return at the end of the file.

Associated files

`lfk.s, lfk.xrel`

Distribution

Applix 1616 Shareware Disk # 15 /lfk

Author

Cameron Hutchison, 9/4/1989.

lisp - ai language - Andrew Morton - SW#8

lisp

Description

Language often used by ai.

Quibbles

I couldn’t persuade it to do squat all.

Associated files

`fact.lsp, fib.lsp, hanoi.lsp, lisp.doc (extensive), lisp.xrel, pp.lsp, queens.lsp, queens2.lsp, readme.txt, simplepp.lsp, starter.bak, starter.doc`, also /lisp/source, /lisp/untested

See also

Distribution

Applix 1616 Shareware Disk #8 /lisp, /lisp/source, /lisp/untested 155 blocks in /lisp

Author

Original version 1.6 by David Betz, Jan 6, 1986.
Conversion to Applix 1616 by Andrew Morton

locate - MRD to find data in memory, by Michael Johnson - SW#5

author	:- MICHAEL JOHNSON, WAGGA WAGGA
program name	:- LOCATE.MRD
program source code	:- LOCATE.S
date	:- 16-8-88
version	:- 1.1
document issue	:- 2

LOCATE MRD

This mrd is used to find out where your data is stored in memory. This will allow you to inspect these variables via the trap #1 (ssasm) or the trap_one() function (C) which use the exception handler. When the trap exception is executed type "LOCATE" from 1616o/s command level and note the addresses given.

It is not essential to use this mrd via the trap #1 (ssasm) trap_one() function (C) but it is advisable. To use the mrd after exit from program do not deactivate the mrd before you exit your program and it will still work.

You can then use the memory dump commands (MDB, MDW, MDL) to inspect these locations. This will give you an idea of what this mrd is capable of particularly with .XREL program files. This mrd will assist in eliminating the need for .EXEC program files. The data structures and call routines can be removed when your software is fully debugged or you may leave them in the code (your choice).

DO NOT leave trap #0 or trap #1 in your code or you might get a nasty surprise!

HOW TO USE THE LOCATE MRD

The way to use the LOCATE mrd is to ensure that your software activates the mrd and that you have also supplied a valid data structure (explained later).

METHOD

1) Run your software and when it finishes type "locate" at 1616o/s command level and it will print your information. NOTE to use this method do not deactivate the locate mrd before you exit your software.

2) Activate the exception handler with the "exception" command. Make sure it says that the MRD vectors are set. IF the 1616o/s vectors are set then repeat the command. Run your software with the trap #1 (ssasm) OR trap_one() (C) inserted into your software at the appropriate locations. When the exception executes you then type "locate" and your information will be printed. NOTE

deactivate the mrd before you exit from your software. When you are finished you then input command "exception" at 1616o/s level and make sure that the 1616o/s vectors are set.

3) Activate the exception mrd and the locate mrd from within your software and simply run your software with the trap #1 (ssasm) OR trap_one() (C) inserted into your software at the appropriate locations. When the exception executes you then type "locate" and your information will be printed. NOTE deactivate both mrd's before you exit.

RECOMMEND method is method 2

You only use method 3 if you are also debugging an address, buss or illegal instruction error.

NOTE The trap_one() function is supplied on this disc in LIBE.LIB in the sub-directory LIBRARY (see documentation of this disc DISC.DOC).

THE MRD COMMAND FUNCTIONS AND USES

RESETS

A reset (all levels) will set the data structure pointer to zero. The enable flag is also set to enabled. Return value is 0.

ENABLE

This command does not require an argument. It will re-enable the mrd if it is disable. Return value is 0.

DISABLE

The MRD is disabled when this command is sent to it. No argument is required. The data structure pointer is set to zero. Return value is 0.

DOIT

The doit command requires an argument which is a pointer to your data structure (explained later). If the mrd is disabled then this will return -1 to indicate that the mrd is disabled. If the pointer you supplied was found to be invalid then it will set it's own pionter and return 1. If your pointer is valid and used then it will return 0.

STOPIT

The MRD is deactivated when this command is sent to it. No argument is required. The data structure pointer is set to zero. The enable flag remains active. Return value is 0 if all is ok IF the mrd is disabled then it will return -1.

VERSION

This command will return the version of the MRD as documented in 1616 technical reference manual. The first MRD version is 1.1 (hopefully this will not need any updating).

NAME

This command will return a pointer to the MRD's name LOCATE as documented in the 1616 technical reference manual.

EXECENTRY

This command will return a pointer to the MRD's name command level entry address as documented in the 1616 technical reference manual.

COMMAND LEVEL USAGE OF THE LOCATE MRD

The LOCATE mrd requires the user to supply a pointer to a data structure within their program. The command level is designed to be used from within the users software via the trap #1 instruction (assembler) or the trap_one() function (C compiler). However it can be used after the program has been exited IF the mrd was not disabled or the stopit command has not be sent by your software. The LOCATE mrd will check to see if the pointer is set, if not it will print "pointer not set". IF the pointer is set but points to a long word (int in C) which is zero then it will print "no data available".

USING THE MRD FROM WITHIN YOUR SOFTWARE (assembler)

The first step is to make sure that the LOCATE MRD is loaded. To do this we look for it by name and at the same time we also find its call value and store it. All of the following code is written to be relocatable without any changes.

First we must find the MRD. If it is not present then we can't use it. The following equates are used in this example.

```
callmrd             equ         89          call mrd syscall
doit   equ          7           doit mrd command
stopit equ          8           stopit mrd command
mrdname             equ         3           find mrd name

* find the LOCATE mrd

       lea          name(pc),a2             get address of name
       move.l       #0,d7       mrd's are zero numbered
       move.l       #mrdname,d2             get name command
tstmrd move.l       d7,d1       load the MRD number
       move.l       #mrdname,d2             command to get the
name pointer
       move.l       #callmrd,d0             call the driver
       trap         #system

       tst.l        d0          found one
       bmi          retrn       no must have been through them
all

* we have a name now test for a match
```

```
       move.l      d0,a0       mrd's name from callmrd
       move.l      a2,a1       name we are looking for
cmptst cmp.b       (a0)+,(a1)+             test character by
character
       bne         next        not this mrd try next
       tst.b       (a0)        end of name
       beq         found       yes we found it
       bra         cmptst      no test next character

next   addq.l      #1,d7       no wrong one try the next
       bra         tstmrd

* name of MRD

name   dc.b        'LOCATE',0,0            locate mrd
```

This section of code is used when the MRD cannot be found. You will need to formulate this section to fit your situation.

```
retrn  ????        code to handle this situation
```

The MRD has been found so now you setup the data structure.

The MRD must be activated. The doit command requires an argument which is a pointer to the data structure.

```
       lea         locate(pc),a0           address of data
structure (argument)
       lea         mrdnum(pc),a1           get address of
number
       move.l      (a1),d1     load the MRD number
       move.l      #doit,d2    command to activate it
       move.l      #callmrd,d0             call the driver
       trap        #system
       tst.l       d0          is it enabled
       bmi         disabled    no
       beq         notset      the pointer was not set (in-
valid)
```

At this point the MRD has been activated and loaded with the information it requires to operate.

BEFORE YOU EXIT BACK TO 1616O/S

IF you have activated the MRD then it is advisable to deactivate the MRD before you exit from your program (not absolutely neccessary but be neat and do it). This is a very simple procedure which requires no argument. The following code will achieve this :-

```
* deactivate the LOCATE mrd and reset the structure pointer
and
* restore 1616o/s vectors in the vectore table
       lea         mrdnum(pc),a1
       move.l      (a1),d1     mrd number
       move.l      #stopit,d2  deactivate will reset the
*                              data structure pointer
       move.l      #callmrd,d0             call the driver
       trap        #system
```

You are able to exit from your program safely at this point.

THE DATA STRUCTURE

```
number equ        3          this many variables in locate
structure

var3   dc.l       0          some variable you have used

numstr dc.b       'mrd driver number',0
locstr dc.b       'address of locate data structure',0
var3st dc.b       'variable var3',0

locate dc.l       number     number of variables to locate
       dc.l       mrdnum     address of the first variable
       dc.l       numstr     address of string description
       dc.l       locate     address of this data structure
       dc.l       locstr     address of string description
       dc.l       var3       ....
       dc.l       var3st     etc.....
```

USING LOCATE MRD FROM C COMPILER

THE DATA STRUCTURE

This structure is an example of the structuer used by the LOCATE mrd it gives the pointers and string labels for use by the mrd. NOTE the structure only contains pointers to the strings and variables.

```
struct data_locate
{
int    number_of_variables;
int    *var1;                              /* variable address
1 */
char   *varst1;                /* pointer to string ident */
       int         *var2;
       char        *varst2;
       int         *var3;
       char        *varst3;
} locate_data;
```

FINDING THE MRD

The LOCATE mrd must be present before we can use it. Therefore we must now look for the mrd. The following is an extract from the function which will be doing the call to find_mrd(). (a later example will put this together).

```
char   *name;
int    mrd_number;             /* uses this mrd */

name = "LOCATE";
mrd_number = findmrd( name ); /* find mrd LOCATE */
```

This function is used to find an mrd by name if it present it will call each mrd in turn starting from zero. if it cannot find the mrd then it will return -1 (a zero would be mrd zero).

```
int    findmrd(name)          /* find an MRD by name */
char   *name;      /* points to mrd's name */
{
char   *mrd_name;  /* pointer returned by get mrd name */
int    mrd_numb = 0;          /* count the MRD's as we go */

/* the while loop will test if this number MRD exists */

while( ( (int) ( mrd_name = get_mrd_name(mrd_numb) ) ) > 0)
{
if ( strcmp(name, mrd_name) )  /* is this the one */

mrd_numb += 1;     /* no try the next */

else
return(mrd_numb);  /* yes return its number */
}

return(-1);        /* MRD not found so say so */
}
```

The next step is to make sure that the MRD was found. If it was not found then you will need to supply the code to suit your situation in this case we have just printed it was not found and then exited to the calling function.

```
if ( !mrd_number ) /* did we find it */
       {
       printf("cannot find %s mrd\n", name);
       return(0);
       }
```

Here we have found the mrd and simply state what the mrd number is. You then setup the locate_data structure with the following code. This data may have been set up previously and from within different functions. I would strongly suggest that you make each variable static when being used by the locate mrd and when you are satisfied with its performance then you can remove the static keyword and make it auto if neccessary.

```
printf("%s is mrd number %d\n", name, mrd_number);

varst1 = "name of mrd in test_locate()";
varst2 = "mrd number in test_locate()";
varst3 = "location of locate_data structure";

locate_data.var1   = (int*) name;
locate_data.varst1 = varst1;
locate_data.var2   = &mrd_number;
locate_data.varst2 = varst2;
locate_data.var3   = (int*) &locate_data;
locate_data.varst3 = varst3;
locate_data.number_of_variables = 3;
```

When all this has been done you simply activate the mrd and supply the ADDRESS of the locate_data structure as an argument.

```
doit_mrd(mrd_number, &locate_data);
```

This enormous function actually activates the locate mrd. The argument value is the address of the locate_data structure.

```
doit_mrd(number, argument)     /* activate the mrd */
int    number, argument;
{
int    command = 7;            /* doit */
return( (int) callmrd(command, number, argument) );  /* do it
*/
}
```

Before you exit it is not essential to deactivate the mrd but to keep things tidy it is a good idea. IF YOU SET IT THEN RESTORE IT !!!!!

AN EXCEPTION TO THIS IS if you are not using the trap_one() function in the exception handler in which case when your program has returned to 1616o/s you can simply type "locate" and it will print your data for you. However, the values of your variables may not be valid.

```
stopit_mrd(mrd_number); /* it's nice to be safe */
```

This function can save a lot of late nights (maybe).

```
stopit_mrd( number )           /* if you can't work it out it's
too LATE go to bed */
int    number;
{
int    command = 8;            /* stopit */
callmrd( command, number, 0);  /* no argument for stopit */
}
```

The following group of functions are used by the previously described functions. They are self explanatory (I hope).

```
char   *get_mrd_name(number)   /* get the pointer to the MRD's
name */
int    number;
{
int    mrd_name = 3;
return( (char *) callmrd( mrd_name, number, 0 ) );
}
void   *callmrd(command, number, argument) /* do this command
with this MRD */
int    command, number, argument;
{
int    callmrd = 89;           /* system call for callmrd */
return( (char *) trap7(callmrd, number, command, argument));
}
```

PUT IT ALL TOGETHER

The following function was used to test the locate mrd. It is the combination of all the parts described above. The file TEST.C in the source directory will give a complete working example of this mrd BUT be carefull it also uses the exception handler.

```
testing()
{
int    mrd;

       mrd = test_locate();
       stopit_mrd(mrd); /* it's nice to be safe */

}

/*     This function will test the LOCATE mrd */

test_locate()
{
char   *name, *varst1, *varst2, *varst3;
int    mrd_number;            /* uses this mrd */

       name = "LOCATE";
       mrd_number = findmrd( name ); /* find mrd LOCATE */

       if ( !mrd_number )
                  {
                  printf("cannot find %s mrd\n", name);
                  return(0);
                  }
       printf("%s is mrd number %d\n", name, mrd_number);

/* now insert the data into the locate_data structure */

       varst1 = "name of mrd in test_locate()";
       varst2 = "mrd number in test_locate()";
       varst3 = "location of locate_data structure";

       locate_data.var1   = (int*) name;
       locate_data.varst1 = varst1;
       locate_data.var2   = &mrd_number;
       locate_data.varst2 = varst2;
       locate_data.var3   = (int*) &locate_data;
       locate_data.varst3 = varst3;
       locate_data.number_of_variables = 3;

       doit_mrd(mrd_number, &locate_data);

       return(mrd_number);

}

/* NEAT AY */
```

I hope you find this program very usefull

KIND REGARDS
MICHAEL JOHNSON

(069) 255255

(I tried for 25 1616 but alas it was not available)

makeega - run Hercules monitor as EGA - Lindsay Washusen - SW#15

makeega

Description

Tweak Conal Walsh’s EGA driver 6545 register values so it runs a Hercules style monochrome monitor in reasonable sytle.

Associated files

`makeega.shell`

Distribution

Applix 1616 Shareware Disk # 15

Author

Lindsay Washusen.

mem - memory free report - Andrew McNamara - SW#15

mem

Description

Reports on the amount of memory free. The report includes the largest block free, the largest allocatable block available, the kludge factor currently in use (see the *Programmers Manual* for details), and the total memory free.

Examples

Quibbles

Associated files

`mem.c, mem.xrel`

See also

Distribution

Applix 1616 Shareware Disk #15 /McNamara

Author

Andrew McNamara

modem - Xmodem file transfer - Mark Harvey - SW#1, SW#2

modem `sa:[sb:]`

Description

Configure the selected serial port to the correct baud rate using the 1616 `serial` command before starting the program. Type **help** pretty much any time to get the menu back.

Menu driven.

Full Full duplex, with no echo of characters.

Half Half duplex, with local echo.

xx Transmit file using xmodem.

yx Transmit file using ymodem

rx Receive file using xmodem.

system Return to operating system.

Alt c Move back one menu level, or stop a transfer.

] Exec an operating system command.

Examples

Quibbles

`xrel` files often get scrambled going to BBS - reason is that any *xmodem* protocol can add up to 128 bytes to the last block of the file, due to the nature of the protocol. Somewhat dated.
Does not strip parity bit in terminal mode. Do not use it to transfer 8 bit files within a 7 bit environment.

Associated files

`modem.exec, modem.mac, modem.s, modem.doc`

See also

rb, **sb**, sealink(?)

Distribution

Applix 1616 Shareware Disk #1 /utilities (V1.1, 25-10-1987), #2 /modem (V1.2, 4-3-1988)

Author

Applix version by Mark Harvey

modem32 - menu driven file transfer program - Sid Young - SW#6

modem32

Description

Demo version of a nice, menu driven modem program, with xmodem protocol, tested with MicroBe Telecom program. Ymodem support is expected in future.

File upload and file download, save buffer to disk.

Selectable com port, data rates, full and half duplex, dial, redial, etc.

Quibbles

Associated files

`modem32.exec`, `modem32.s`, `modem32.doc`

See also

modem

Distribution

Applix 1616 Shareware Disk # 6/utilities/sid

Author

Sid Young

more - formats text files for viewing - Matthew Gardner - SW#15

more `[ files ... ]`

Description

More is a filter which allows examination of continuous text, one screenful at a time, on a display. It knows about wildcards and pathnames. It is retained in UNIX for backward compatibility (there are better versions, such as less). This version has more features than the C version from Andrew Morton.

If no filename is given, `more` assumes it is reading from a pipe, or standard input. Use whatever your EOF character is (Ctrl d, Ctrl z) to exit in this case.

It pauses after each screenful, printing -More- at the bottom of the screen. If the user type Space, Enter, PgDn or almost any other character, another screenful is displayed. Use it instead of `type` or `edit` for reading a file. Other characters used interactively include:

b, **B**, PgUp	Backwards a page, to see previous page.
h, **H**	Help, a list of available commands is shown.
q, **Q**	Quit to next file, exit from **more** if last file.
?	Help, a list of available commands.
Esc	Exit from `more`.
/	pattern. Search forward for `pattern`. A lower case pattern will find both upper and lower case matches. A pattern starting with Upper case will find both upper and lower case matches. An UPPER case only pattern will find only an UPPER case match.
n, **N**	Next occurrence of previous pattern.

Examples

A sample usage of **more** (in UNIX) in previewing **roff** output would be

```
roff -s +2 doc.n | more
```

Bugs

When first invoked, the first line of the file is lost off the top of the display.

Associated files

`more.s`

See also

cat, **cio**, **pr**, **roff**, **more.c**

Distribution

Applix 1616 Shareware #15 /gardner

Author

Matthew Gardner, Box 8021, Palmerston North, New Zealand

nswp32 - bulk file copy, delete, view utility - Sid Young - SW#6

nswp

Description

A menu driven file copy, delete, and view utility, intended to handle files in bulk by "tagging" those to be acted upon. Probably similar to CP/M **sweep** utility. Looks like an excellent starting point for a full "tree" style file manager.

- [a] Retag a file.
- [b] Back one file.
- [c] Copy a file.
- [d] Delete a file.
- [f] Find a file (not yet implemented).
- [h] Help.
- [l] Log new drive.
- [m] Mass copy tagged files.
- [r] Rename a file.
- [s] Status display.
- [t] Tag a file.
- [u] Untag a file.
- [v] View contents of file.
- [x] E**x**it from view or program.
- [/] To next directory.
- [\] Back one directory.

Quibbles

The present version is fairly slow in operation.
Had problems with the display of directories, perhaps due to sub-directory prompts.
Bombed on exit, had to do a power down to recover.
I’m not sure Sid’s hard coded approach is wise, given the rapidity of changes to 1616/OS. Changes probably wouldn’t affect it as much if the directory information were stored in strings, and passed direct to Andrew’s routines.

Associated files

`nswp.xrel`, `nswp.s`

See also

ftree

Distribution

Applix 1616 Shareware Disk # 6/utility/sid

Author

Sid Young
Suggested donation $5

pfile - menu driven flat file data base - Paul Cahill - SW#4

pfile.bas

Description

Menu driven flat file data base, written in SSBASIC, using disk files. Allows viewing of available file names, creation of new data bases, viewing of all data, or search for specific information.

Printer output is allowed, including output of specific records as they are located (very handy).

One very nice feature is that Paul has provided a large range of electronic and computing related data files, mainly relating to *Electronics Australia*, from 1976 to August 1988. I expect I'll be spending far less time searching through back issues as a result of Paul's efforts in entering this information.

Quibbles

Associated files

```
pfile.bas
1616owners.f, ea-cdi80.f, ea-errata.f, ea-index70.f,
ea-index80.f, ea-microconst80.f, ea-pc80.f, ea-theory.f,
ea-tvs.f
```

See also

grep, **sort**

Distribution

Applix 1616 Shareware Disk # 4/database

Author

Paul Cahill, 65 Labrador Street, Rooty Hill, NSW 2766

prog?

Description

prog1	Add 48 to a set of numeric values held as data, print the ASCII equivalent.
prog2	Load text into a buffer, then display it.
prog3	Sorts and prints valid ASCII codes
prog4	Parameter passing program, prints a message.
prog5	Passes a message via the stack.
prog6	Analyses petrol consumption and mileage figues of data declared in program.
prog7	Sorts a set of ASCII characters alphabetically.

Quibbles

The conversion to Applix isn't always elegant in these tutorials ... on the other hand, if it were, it might be too different to the book to allow novices to follow it. Catch 22.

Associated files

See also

`prog?.s`

Distribution

Applix 1616 Shareware Disk #1 /tutorial

Author

Original version by Robert Erskine, in *First Steps in Assembly Language*

Conversion to Applix 1616 by Kathy Morton

ps3 - process status nice value - Matthew Geier or Andrew McNamara - SW#15

ps3

Description

A variation on the usual `ps` command, this gives a brief `ps` which includes the ***nice*** value.

Associated files

`ps3.c`

Distribution

Applix 1616 Shareware Disk # 15 /geier /McNamara

Author

Matthew Geier or Andrew McNamara, both from an Andrew Morton original.

quick - mrd to speed up processing - Andrew Morton - SW#9

quick.mrd

Description

A memory resident driver, intended to speed up processing in 15 MHz models by shutting down most of the excess overhead associated with video refresh. This cuts out the 4 wait states when running video, by not actually displaying more than a few lines.

quick on — Fast processing, 4 lines displayed.

quick off — Normal display, slowest processing.

quick *nn* — Number of display rows, 2 to 24.

quick enable

quick disable

Alt Shift Q

Alt Shift U

Alt Shift D

Examples

Quibbles

Associated files

`quick.s, quick.mrd`

See also

dhrystone

Distribution

Applix 1616 Shareware Disk #9 /15mhz

Author

Andrew Morton

quindex - quick index - Michael Johnson - SW#5

quindex

Description

Menu driven quick index program, from which to launch other executable programs. Designed to provide a user friendly shell for people playing games, etc. Very elegant and attractive appearance. You set it up with a set of descriptive names, filepaths, and filenames, for the programs it can display. Intended more for hard disk users, but should run from floppy also.

It expects to find a datafile in `/sysdev/quindex.file`.

It has four modes of operation, selected via the PgUp and PgDn keys, while Esc exits to 1616/OS. The F1 key makers a temporary exit to 1616/OS (return by using `quit`).

Execute Use arrow keys to select an entry, Enter key to execute the selected program. It will return to `quindex` after exit from the program. Home goes to the first entry, End takes you to the last entry.

Edit Change the displayed name, directory, or executable command. Use cursor keys to move, Enter to slect an item for editing.

Delete Get rid of an entry.

Insert Displays boxes for entry of the name (14 characters maximum), directory, and executable command line to be sent to 1616/OS. If last entry in this is **[** you can add parameters prior to execution.

Examples

Quibbles

Every time I write this up, Michael adds more features. This is now a very powerful menu system, which will make it easy for those unfamiliar with computers to use programs.

Associated files

```
ccquindex.shell, quindex.c, quindex.h, quindex.obj, quin-
dex.xrel, quindex1.c, quindex.obj
```

See also

ftree

Distribution

Applix 1616 Shareware Disk #5 /quindex

Author

Suggested donation $5. Michael Johnson

rb - receive yam or ymodem files - Colin McCormack - SW#3

rb [-7buv] [-sA] [-sB]
rb [-1bcuv] filename

Description

Receives binary files over serial port from a computer running YAM or ymodem, and places the file in `filename`. Supports CRC or checksum. Log of activities is maintained in `rblog`.

-7 mask is octal 0177

-b binary option, with **^Z** as end of file

-c CRC (cyclic redundancy check (also **-k**)

-q quiet mode, see also **-v**

-s serial port A or B

-t text mode, don't receive binaries

-u pathname treated as upper case

-v verbose mode, to `rblog`

Quibbles

Under 1616/OS V2 and early V3, would bomb on seek to a zero length file (its own log file).

Associated files

`rb.exec, rb.c`

See also

sb

Distribution

Applix 1616 Shareware Disk #3 /utilities/ymodem

Author

Chuck Forsberg, from UNIX `yam2` and `sb`

Conversion to Applix 1616 by Colin McCormack

readmrd - reports on MRDs - Michael Johnson - SW#3, SW#5

readmrd

Description

Locates and reports on all MRDs present in memory. Gives address, version number, and the name of each such MRD.

Quibbles

Clears screen prior to display, which can be annoying.

The hard disk MRD seems to have a bug, and the system will crash if you use this program when the hard disk MRD is installed.

It would be nice to display the *state* of an MRD with this software, but alas Andrew has not allowed for this possibility.

Associated files

`readmrd.exec, readmrd.s`

Distribution

Applix 1616 Shareware Disk #3 /

Author

Michael Johnson, 11/12 Kokoda St. Wagga Wagga NSW 2650 (069) 255255 (home,) (069) 230388 (work - a free call).
Suggested donation $10.

Readmrd documentation by Michael Johnson

author:- MICHAEL JOHNSON, WAGGA WAGGA
program name:- READMRD.XREL
program source code:- READMRD.S
date:- 16-8-88
version :- 1.0
document issue :- 1

INTRODUCTION

This program is used to find which mrd's are currently loaded onto the system. It will also tell you the execute entry address and the version number. The mrd's are listed in order i.e. the first one listed is mrd 0, the next is mrd 1, etc.. This program is of use when you wish to some software which uses an mrd and you don't know if the mrd is loaded.

USAGE

The command `READMRD` will execute this program. There are no parameters required by this program. The result printout is as follows :-

```
/JOHNSON>readmrd

execute         version         name

00054BF40       1.2             TDOS
00054D120       1.2             ASSIGN
000559880       1.1             LOCATE
00055C7A0       2.1             EXCEPTION
```

I am not sure of exactly how useful this program is even though I do find it handy occasionally. If you find it particularly useful or would like to see some other software of this nature PLEASE tell me.

I bet you never knew you could log onto a device (/f0) by it's volume name. Try it. You may not be sure which drive but you will know the volume.

sb - send yam or ymodem files - Colin McCormack - SW#6

sb [-7dfkquv][-sA][-sB] filename ...

Description

Sends one or more binary files over serial port to a computer running YAM or ymodem, from the file or files in `filename`. Supports CRC or checksum. Maintains a logfile in `sblog`.

-7 mask is octal 0177

-d dot to slash conversion of names

-f full names transmitted

-k 1k record lengths used

-q quiet mode, see also **-v**

-s serial port A or B

-u unlink after operation

-v verbose mode, to `rblog`

Quibbles

Associated `rb` bombs on seeks to zero length files (its own log file) under 1616/OS V2 and early V3.

Associated files

`sb.exec, sb.c`

See also

rb

Distribution

Applix 1616 Shareware Disk #3 /utilities/ymodem

Author

Chuck Forsberg, from UNIX `yam2` and `sb`

Conversion to Applix 1616 by Colin McCormack

sc - spreadsheet calculator - Andrew Morton - SW#15

sc [-c][-m][-n][-r][-x][file]

Description

The spreadsheet calculator sc is based on rectangular tables much like a financial spreadsheet. When invoked it presents you with a table organized as rows and columns of cells. If invoked without a file argument, the table is initially empty. Otherwise file is read in (see the Get command below). Each cell may have associated with it a numeric value, a label string, and/or an expression (formula) which evaluates to a numeric value or label string, often based on other cell values.

Options are:

-c Start the program with the recalculation being done in column order.

-m Start the program with automatic recalculation disabled. The spreadsheet will be recalculated only when the ‘‘@’’ command is used.

-n Start the program in quick numeric entry mode (see below).

-r Start the program with the recalculation being done in row order (default option).

-x Cause the Get and Put commands (see below) to encrypt and decrypt data files.

All of these options can be changed with the ^T and S commands (see below) while sc is running. Options specified when sc is invoked override options saved in the data file.

General Information

The screen is divided into four regions. The top line is for entering commands and displaying cell values. The second line is for messages from sc. The third line and the first four columns show the column and row numbers, from which are derived cell addresses, e.g. A0 for the cell in column A, row 0. Note that column names are case-insensitive: you can enter A0 or a0.

The rest of the screen forms a window looking at a portion of the table. The total number of display rows and columns available, hence the number of table rows and columns displayed, is set by curses(3) and may be overridden by setting the lines and columns environment variables, respectively.

The screen has two cursors: a cell cursor, indicated by a highlighted cell and a < on the screen, and a character cursor, indicated by the terminal's hardware cursor. The cell and character cursors are often the same. They differ when you type a command on the top line.

If a cell's numeric value is wider than the column width (see the f command), the cell is filled with asterisks. If a cell's label string is wider than the column width, it is truncated at the start of the next non-blank cell in the row, if any.

Cursor control commands and row and column commands can be prefixed by a numeric argument which indicates how many times the command is to be executed. You can type ^U before a repeat count if quick numeric entry mode is enabled or if the number is to be entered while the character cursor is on the top line.

Commands which use the terminal's control key, such as ^N, work both when a command is being typed and when in normal mode.

Changing Options

^To Toggle options. This command allows you to switch the state of one option selected by o. A small menu lists the choices for o when you type ^T. The options selected are saved when the data and formulas are saved so that you will have the same setup next time you enter the spreadsheet.

a Automatic Recalculation. When set, each change in the spreadsheet causes the entire spreadsheet be recalculated. Normally this is not noticeable, but for very large spreadsheets, it may be faster to clear automatic recalculation mode and update the spreadsheet via explicit **@** commands. Default is automatic recalculation on.

c Current cell highlighting. If enabled, the current cell is highlighted (using the terminal's standout mode, if available) in addition to being marked by the cell cursor.

e External function execution. When disabled, external functions (see @ext() below) are not called. This saves a lot of time at each screen update. External functions are disabled by default. If disabled, and external functions are used anywhere, a warning is printed each time the screen is updated, and the result of @ext() is the value from the previous call, if any, or a null string.

n Quick numeric entry. If enabled, a typed digit is assumed to be the start of a numeric value for the current cell, not a repeat count, unless preceded by ^U.

t Top line display. If enabled, the name and value of the current cell is displayed on the top line. If there is an associated label string, the first character of the string value is < for a leftstring or > for a rightstring

(see below), followed by "string" for a constant string or {expr} for a string expression. If the cell has a numeric value, it follows as [value], which may be a constant or expression.

x Encryption. See the -x option.

$ Dollar prescale. If enabled, all numeric constants (not expressions) which you enter are multipled by 0.01 so you don't have to keep typing the decimal point if you enter lots of dollar figures.

S Set options. This command allows you to set various options. A small menu lists the options that cannot be changed through ^T above.

byrows/bycols
Specify the order cell evaluation when updating. These options also affect the order in which cells are filled (see /f) and whether a row or column is cleared by an x command.

iterations= n
Set the maximum number of recalculations before the screen is displayed again. Iterations is set to 10 by default.

tblstyle= s
Control the output of the T command. s can be: 0000 (default) to give colon delimited fields, with no tbl control lines; tbl to give colon delimited fields, with tbl(1) control lines; latex to give a LaTeX tabular environment; and tex to give a TeX simple tabbed alignment with ampersands as delimiters.

Other Set options are normally used only in sc data files since they are available through ^T. You can also use them interactively.

autocalc/!autocalc	Set/clear auto recalculation mode.
numeric/!numeric	Set/clear numeric mode.
prescale/!prescale	Set/clear numeric prescale mode.
extfun/!extfun	Enable/disable external functions.
cellcur/!cellcur	Set/clear current cell highlighting mode.
toprow/!toprow	Set/clear top row display mode.

Cursor Control Commands

^P Move the cell cursor up to the previous row.

^N Move the cell cursor down to the next row.

^B Move the cell cursor backward one column.

^F Move the cell cursor forward one column.

h, j, k, l
If the character cursor is not on the top line, these are alternate, vi-compatible cell cursor controls (left, down, up, right).

^H If the character cursor is not on the top line, ^H is the same as ^B.

SPACE
: If the character cursor is not on the top line, the space bar is the same as ^F.

TAB
: If the character cursor is on the top line, TAB starts a range (see below). Otherwise, it is the same as ^F.

Arrow Keys
: The terminal's arrow keys provide another alternate set of cell cursor controls if they exist and are supported in the appropriate `termcap` entry. Some terminals have arrow keys which conflict with other control key codes.

  For example, a terminal might send ^H when the back arrow key is pressed. In these cases, the conflicting arrow key performs the same function as the key combination it mimics.

^ Move the cell cursor up to row 0 of the current column.

Move the cell cursor down to the last valid row of the current column.

0 Move the cell cursor backward to column A of the current row. This command must be prefixed with ^U if quick numeric entry mode is enabled.

$ Move the cell cursor forward to the last valid column of the current row.

b Scan the cursor backward (left and up) to the previous valid cell.

w Scan the cursor forward (right and down) to the next valid cell.

^E dGo to end of range. Follow ^E by a direction indicator such as ^P or j.
: If the cell cursor starts on a non- blank cell, it goes in the indicated direction until the last non-blank adjacent cell. If the cell cursor starts on a blank cell, it goes in the indicated direction until the first non-blank cell. This command is useful when specifying ranges of adjacent cells (see below), especially when the range is bigger than the visible window.

g Go to a cell. `sc` prompts for a cell's name, a regular expression surrounded by quotes, or a number. If a cell's name such as `ae122` or the name of a defined range is given, the cell cursor goes directly to that cell. If a quoted regular expression such as `"Tax Table"` or `"^Jan [0-9]*$"` is given, `sc` searches for a cell containing a string matching the regular expression. See `regex(3)` or `ed(1)` for more details on the form of regular expressions. If a number is given, `sc` will search for a cell containing that number. Searches for either strings or numbers proceed forward from the current cell, wrapping back to a0

at the end of the table, and terminate at the current cell if the string or number is not found. The last g command is saved, and can be re-issued by entering g Enter.

Cell Entry and Editing Commands

Cells can contain both a numeric value and a string value. Either value can be the result of an expression, but not both at once, i.e. each cell can have only one expression associated with it. Entering a valid numeric expression alters the cell's previous numeric value, if any, and replaces the cell's previous string expression, if any, leaving only the previously computed constant label string. Likewise, entering a valid string expression alters the cell's the previous label string, if any, and replaces the cell's previous numeric expression, if any, leaving only the previously computed constant numeric value.

= Enter a numeric constant or expression into the current cell. sc prompts for the expression on the top line. The usual way to enter a number into a cell is to type =, then enter the number in response to the prompt on the top line. The quick numeric entry option, enabled through the -n option or ^T command, shows the prompt when you enter the first digit of a number (you can skip typing =).

< Enter a label string into the current cell to be flushed left against the left edge of the cell.

" > Enter a label string into the current cell to be flushed right against the right edge of the cell.

Strings you enter must start with ". You can leave off the trailing " and sc will add it for you. You can also enter a string expression by back-spacing over the opening " in the prompt.

e Edit the value associated with the current cell. This is identical to = except that the command line starts out containing the old numeric value or expression associated with the cell.

E Edit the string associated with the current cell. This is identical to <, ", or > except that the command line starts out containing the old string value or expression associated with the cell.

To enter and edit a cell's number part, use the = and e commands. To enter and edit a cell's string part, use the <, ", >, and E commands. See the sections below on numeric and string expressions for more information.

x Clear the current cell. Deletes the numeric value, label string, and/or numeric or string expression. You can prefix this command with a count of the number of cells on the current row to clear. The current column is used if column recalculation order is set. Cells cleared with this command may be recalled with any of the pull commands (see below).

m Mark a cell to be used as the source for the copy command.

c Copy the last marked cell to the current cell, updating row and column references in its numeric or string expression, if any.

+ If not in numeric mode, add the current numeric argument (default 1) to the value of the current cell. In numeric mode, + introduces a new numeric expression or value, the same as =.

- If not in numeric mode, subtract the current numeric argument (default 1) from the value of the current cell. In numeric mode, - introduces a new, negative, numeric expression or value, like =.

File Commands

G Get a new database from a file. If encryption is enabled, the file is decrypted before it is loaded into the spreadsheet.

P Put the current database into a file. If encryption is enabled, the file is encrypted before it is saved.

W Write a listing of the current database into a file in a form that matches its appearance on the screen. This differs from the Put command in that its files are intended to be reloaded with Get, while Write produces a file for people to look at. Hidden rows or columns are not shown when the data is printed.

T Write a listing of the current database to a file, but include delimiters suitable for processing by the tbl, LaTeX, or TeX table processors. The delimiters are controlled by the tbl style option. See Set above. The delimters are are a colon (:) for style 0 or tbl and an ampersand (&) for style latex or tex.

With the Put, Write, and Table commands, the optional range argument writes a subset of the spreadsheet to the output file.

With the Write and Table commands, if you try to write to the last file used with the Get or Put commands, or the file specified on the command line when sc was invoked, you are asked to confirm that the (potentially) dangerous operation is really what you want.

The three output commands, Put, Write, and Table, can pipe their (unencrypted only) output to a program. To use this feature, enter | program to the prompt asking for a filename. For example, to redirect the output of the Write command to the printer, you might enter | lpr -p.

M Merge the database from the named file into the current database. Values and expressions defined in the named file are read into the current spreadsheet overwriting the existing entries at matching cell locations.

R Run macros. Since sc files are saved as ASCII files, it is possible to use them as primitive macro definition files. The Run command makes this easier. It's like the Merge command, but prints a saved path name as the start of the filename to merge in. The string to use is set with the Define command. To write macros, you must be familiar with the file format written by the Put command. This facility is still primitive and could be much improved.

D Define a path for the Run command to use.

All file operations take a filename as the first argument to the prompt on the top line. The prompt supplies a " to aid in typing in the filename. The filename can also be obtained from a cell's label string or string expression. In this case, delete the leading " with the backspace key and enter a cell name such as a22 instead. If the resulting string starts with |, the rest of the string is interpreted as a UNIX command, as above.

Row and Column Commands

These commands can be used on either rows or columns. The second letter of the command is either a row designator (one of the characters r, ^ B, ^ F, h, l) or a column designator (one of c, ^ P, ^ N, k, j). A small menu lists the choices for the second letter when you type the first letter of one of these commands. Commands which move or copy cells also modify the row and column references in affected cell expressions. The references may be frozen by using the fixed operator or using the $ character in the reference to the cell (see below).

ir, ic Insert a new row (column) by moving the row (column) containing the cell cursor, and all following rows (columns), down (right) one row (column). The new row (column) is empty.

ar,ac Append a new row (column) immediately following the current row (column). It is initialized as a copy of the current one.

dr, dc Delete the current row (column).

pr, pc, pm Pull deleted rows (columns) back into the spreadsheet. The last deleted set of cells is put back into the spreadsheet at the current location. pr inserts enough rows to hold the data. pc inserts enough columns to hold the data. pm (merge) does not insert rows or columns; it overwrites the cells beginning at the current cell cursor location.

vr, vc Remove expressions from the affected rows (columns), leaving only the values which were in the cells before the command was executed.

zr, zc Hide (''zap'') the current row (column). This keeps a row (column) from being displayed but keeps it in the data base. The status of the rows and columns is saved with the data base so

hidden rows and columns will be still be hidden when you reload the spreadsheet. Hidden rows or columns are not printed by the W command.

sr, sc — Show hidden rows (columns). Enter a range of rows (columns) to be revealed. The default is the first range of rows (columns) currently hidden. This command ignores the repeat count, if any.

f — Set the output format to be used for printing the numeric values in each cell in the current column. Enter two numbers: the total width in characters of the column, and the number of digits to follow decimal points. Values are rounded off to the least significant digit displayed. The total column width affects displays of strings as well as numbers. A preceding count can be used to affect more than one column. This command has only a column version (no second letter).

Range Commands

Range operations affect a rectangular region on the screen defined by the upper left and lower right cells in the region. All of the commands in this class start with a slash; the second letter of the command indicates which command. A small menu lists the choices for the second letter when you type /. sc prompts for needed parameters for each command. Phrases surrounded by square brackets in the prompt are informational only and may be erased with the backspace key.

Prompts requesting variable names may be satisfied with either an explicit variable name, such as A10, or with a variable name previously defined in a /d command (see below). Range name prompts require either an explicit range such as A10: B20, or a range name previously defined with a /d command. A default range shown in the second line is used if you omit the range from the command or press the TAB key (see below). The default range can be changed by moving the cell cursor via the control commands (^P, ^N, ^B, ^F) or the arrow keys. The cells in the default range are highlighted (using the terminal's standout mode, if available).

/x — Clear a range. Cells cleared with this command may be recalled with any of the pull commands.

/v — Values only. This command removes the expressions from a range of cells, leaving just the values of the expressions.

/c — Copy a source range to a destination range. The source and destination may be different sizes. The result is always one or more full copies of the source. Copying a row to a row yields a row. Copying a column to a column yields a column. Copying a range to anything yields a range. Copying a row to a column or a column to a row yields a range with as many copies of the source as there are cells in the

destination. This command can be used to duplicate a cell through an arbitrary range by making the source a single cell range such as b20: b20.

/f Fill a range with constant values starting with a given value and increasing by a given increment. Each row is filled before moving on to the next row if row order recalculation is set. Column order fills each column in the range before moving on to the next column. The start and increment numbers may be positive or negative. To fill all cells with the same value, give an increment of zero.

/d Use this command to assign a symbolic name to a single cell or a rectangular range of cells on the screen. The parameters are the name, surrounded by "", and either a single cell name such as A10 or a range such as a1: b20. Names defined in this fashion are used by the program in future prompts, may be entered in response to prompts requesting a cell or range name, and are saved when the spreadsheet is saved with the Put command. Names defined must be more than two alpha characters long to differentiate them from a column names, and must not have embedded special characters. Names may include the character _ or numerals as long as they occur after the first three alpha characters.

/s This command lists (shows) the currently defined range names. If there are no defined range names, then a message is given, otherwise it pipes output to `sort`, then to less. If the environment variable PAGER is set, its value is used in place of `less`.

/u Use this command to undefine a previously defined range name.

Miscellaneous Commands

Q q ^C
Exit from sc. If you made any changes since the last Get or Put, sc asks about saving your data before exiting.

^G ESC
Abort entry of the current command.

? Enter an interactive help facility. Lets you look up brief summaries of the main features of the program. The help facility is structured like this manual page so it is easy to find more information on a particular topic.

! Shell escape. sc prompts for a shell command to run. End the command line with the RETURN key. If the environment variable SHELL is defined, that shell is run. If not, /bin/sh is used. Giving a null command line starts the shell in interactive mode. A second ‘‘!’’ repeats the previous command.

^L Redraw the screen.

^R Redraw the screen with special highlighting of cells to be filled in. This is useful for finding values you need to provide or update in a form with which you aren't familiar or of which you have forgotten the details.

It's also useful for checking a form you are creating. All cells which contain constant numeric values (not the result of a numeric expression) are highlighted temporarily, until the next screen change, however minor. To avoid ambiguity, the current range (if any) and current cell are not highlighted.

^X This command is similar to ^R, but highlights cells which have expressions. It also displays the expressions in the highlighted cells as left-flushed strings, instead of the numeric values and/or label strings of those cells. This command makes it easier to check expressions, at least when they fit in their cells or the following cell(s) are blank so the expressions can slop over (like label strings). In the latter case, the slop over is not cleared on the next screen update, so you may want to type ^L after the ^X in order to clean up the screen.

@ Recalculates the spreadsheet.

^V Type, in the command line, the name of the current cell (the one at the cell cursor). This is useful when entering expressions which refer to other cells in the table.

^W Type, in the command line, the expression attached to the current cell. If there is none, the result is ''?''.

^A Type, in the command line, the numeric value of the current cell, if any.

The ^V, ^W, and ^A commands only work when the character cursor is on the command line and beyond the first character.

TAB

When the character cursor is on the top line, defines a range of cells via the cursor control commands or the arrow keys. The range is highlighted, starts at the cell where you typed TAB, and continues through the current cell cursor. Pressing TAB again causes the highlighted range to be entered into the command line and the highlighting to be turned off. This is most useful for defining ranges to functions such as @sum(). Pressing ''('' acts just like typing the TAB key the second time and adds the closing ''('' . Note that when you give a range command, you don't need to press the first TAB to begin defining a range starting with the current cell.

Variable Names

Normally, a variable name is just the name of a cell, such as K20. The value is the numeric or string value of the cell, according to context.

When a cell's expression (formula) is copied to another location via copy or range-copy, variable references are by default offset by the amount the formula moved. This allows the new formula to work on new data. If cell references are not to change, you can either use the fixed operator (see below), or one of the following variations on the cell name.

K20 References cell K20; the reference changes when the formula is copied.

K20 Always refers to cell K20; the reference stays fixed when the formula is copied.

$K20 Keeps the column fixed at column K; the row is free to vary.

K$20 Similarly, this fixes the row and allows the column to vary.

These conventions also hold on defined ranges. Range references vary when formulas containing them are copied. If the range is defined with fixed variable references, the references do not change.

fixed To make a variable not change automatically when a cell moves, put the word fixed in front of the reference, for example: B1 * fixed C3.

Numeric Expressions

Numeric expressions used with the = and e commands have a fairly conventional syntax. Terms may be constants, variable names, parenthesized expressions, and negated terms. Ranges may be operated upon with range functions such as sum (@sum()) and average (@avg()). Terms may be combined using binary operators.

-e Negation.

e+e Addition.

e-e Subtraction.

e*e Multiplication.

e/e Division.

e1%e2 e1 mod e2.

e^e Exponentiation.

e<e e<=e e=e e!=e e>=e e>e
Relationals: true (1) if and only if the indicated relation holds, else false (0). Note that ‘‘<=’’, ‘‘!=’’, and ‘‘>=’’ are converted to their ‘‘~()’’ equivalents.

~e Boolean operator NOT.

e&e Boolean operator AND.

e|e Boolean operator OR.

e?e:e Conditional: If the first expression is true then the value of the second is returned, otherwise the value of the third.

Operator precedence from highest to lowest is:

-, ~ ^ *, / +, - <, <=, =, !=, >=, > & | ?:

Built-in Range Functions

These functions return numeric values.

@sum(r) Sum all valid (nonblank) entries in the region whose two corners are defined by the two variable names (e.g. c5:e14) or the range name specified.

@prod(r) Multiply together all valid (nonblank) entries in the specified region.

@avg(r) Average all valid (nonblank) entries in the specified region.

@max(r) Return the maximum value in the specified region. See also the multi argument version of @max below.

@min(r) Return the minimum value in the specified region. See also the multi argument version of @min below.

@stddev(r) Return the sample standard deviation of the cells in the specified region.

@lookup(e,r) @lookup(se,r) Evaluates the expression then searches through the range r for a matching value. The range should be either a single row or a single column. The expression can be either a string expression or a numeric expression. If it is a numeric expression, the range is searched for the the last value less than or equal to e. If the expression is a string expression, the string portions of the cells in the range are searched for an exact string match. The value returned is the numeric value from the next row and the same column as the match, if the range was a single row, or the value from the next column and the same row as the match if the range was a single column.

@index(e,r) Use the value of the expression e to index into the range r. The numeric value at that position is returned. The value 1 selects the first item in the range, 2 selects the second item, etc. R should be either a single row or a single column.

@stindex(e,r) Use the value of e to index into the range r. The string value at that position is returned. The value 1 selects the first item in the range, 2 selects the second item, etc. The range should be either a single row or a single column.

Built-in Numeric Functions

All of these functions operate on floating point numbers (doubles) and return numeric values. Most of them are standard system functions more fully described in `math(3)`. The trig functions operate with angles in radians.

@sqrt(e) Return the square root of e.

@exp(e) Return the exponential function of e.

@ln(e) Return the natural logarithm of e.

@log(e) Return the base 10 logarithm of e.

@floor(e) Return the largest integer not greater than e.

@ceil(e) Return the smallest integer not less than e.

@rnd(e) Round e to the nearest integer.

@fabs(e) Return the absolute value of e.

@pow(e1,e2) Return e1 raised to the power of e2.

@hypot(e1,e2) Return sqrt(e1*e1+e2*e2), taking precautions against unwarranted overflows.

pi A constant quite close to pi.

@dtr(e) Convert e in degrees to radians.

@rtd(e) Convert e in radians to degrees.

@sin(e) @cos(e) @tan(e)
Return trigonometric functions of radian arguments. The magnitude of the arguments are not checked to assure meaningful results.

@asin(e) Return the arc sine of e in the range -pi/2 to pi/2.

@acos(e) Return the arc cosine of e in the range 0 to pi.

@atan(e) Return the arc tangent of e in the range -pi/2 to pi/2.

@atan2(e1,e2) Returns the arc tangent of e1/e2 in the range -pi to pi.

@max(e1,e2,...)
Return the maximum of the values of the expressions. Two or more expressions may be specified. See also the range version of @max above.

@min(e1,e2,...)
Return the minimum of the values of the expressions. Two or more expressions may be specified. See also the range version of @min above.

@ston(se) Convert string expression se to a numeric value.

@eqs(se1,se2) Return 1 if string expression se1 has the same value as string expression se2, 0 otherwise.

@nval(se,e) Return the numeric value of a cell selected by name. String expression se must evaluate to a column name (‘‘A’’-‘‘AE’’) and e must evaluate to a row number (0-199). If se or e is out of bounds, or the cell has no numeric value, the result is 0. You can use this for simple table lookups. Be sure the table doesn’t move unexpectedly! See also @sval() below.

String Expressions

String expressions are made up of constant strings (characters surrounded by double quotation marks), variables (cell names, which refer to the cells’s label strings or expressions), and string functions. Note that string expressions are only allowed when entering a cell’s label string, not its numeric part. Also note that string expression results may be left or right flushed, according to the type of the cell’s string label.

Concatenate strings. For example, the string expression A0 # "zy dog" displays the string ‘‘the lazy dog’’ in the cell if the value of A0’s string is ‘‘the la’’.

Built-in String Functions

@substrr(se,e1,e2)
Extract and return from string expression se the substring indexed by character number e1 through character number e2 (defaults to the size of se if beyond the end of it). If e1 is less than 1 or greater than e2, the result is the null string. For example, `@substr ("Nice jacket", 4, 7)` returns the string ‘‘e jac’’.

@fmt(se,e) Convert a number to a string. The argument se must be a valid printf(3) format string. e is converted according to the standard rules. For example, the expression `@fmt ("**%6.3f**", 10.5)` yields the string ‘‘**10.500**’’. e is a double, so applicable formats are e, E, f, g, and G. Try ‘‘%g’’ as a starting point.

@sval(se,e) Return the string value of a cell selected by name. String expression se must evaluate to a column name (‘‘A’’-‘‘AE’’) and e must evaluate to a row number (0- 199). If se or e is out of bounds, or the cell has no string value, the result is the null string. You can use this for simple table lookups. Be sure the table doesn’t move unexpectedly!

@ext(se,e) Call an external function (program or script). The purpose is to allow arbitrary functions on values, e.g. table lookups and interpolations. String expression se is a command or command line to call with popen(3). The value of e is converted to a string and appended to the command line as an argument. The result of @ext() is a string: the first line printed to standard output by the command. The command should emit exactly one output line. Additional output, or output to standard error, messes up the screen. @ext() returns a null string and prints an appropriate warning if external functions are disabled, se is null, or the attempt to run the command fails.

External functions can be slow to run, and if enabled are called at each screen update, so they are disabled by default. You can enable them with ^T when you really want them called.

A simple example:

```
@ext ("echo", a1)
```

You can use @ston() to convert the @ext() result back to a number. For example:

```
@ston (@ext ("form.sc.ext", a9 + b9))
```

Note that you can built a command line (including more argument values) from a string expression with concatenation. You can also "hide" the second argument by ending the command line (first argument) with ‘‘ #’’ (shell comment).

Built-in Financial Functions

Financial functions compute the mortgage (or loan) payment, future value, and the present value functions. Each accepts three arguments, an amount, a rate of interest (per period), and the number of periods. These functions are the same as those commonly found in other spreadsheets and financial calculators

@pmt(e1,e2,e3)
: @pmt(60000,.01,360) computes the monthly payments for a $60000 mortgage at 12% annual interest (.01 per month) for 30 years (360 months).

@fv(e1,e2,e3)
: @fv(100,.005,36) computes the future value for of 36 monthly payments of $100 at 6% interest (.005 per month). It answers the question: "How much will I have in 2 years if I deposit $100 per month in a savings account paying 6% interest compounded monthly?"

@pv(e1,e2,e3)

@pv(1000,.015,36) computes the present value of an ordinary annuity of 36 monthly payments of $1000 at 18% annual interest. It answers the question: "How much can I borrow at 18% for 3 years if I pay $1000 per month?"

Built-in Date and Time Functions

Time for sc follows the system standard: the number of seconds since 1970. All date and time functions except @date() return numbers, not strings.

@now — Return the current time encoded as the number of seconds since December 31, 1969, midnight, GMT.

The following functions take the time in seconds (e.g. from @now) as an argument and return the specified value. The functions all convert from GMT to local time.

@date(e) — Convert the time in seconds to a date string 24 characters long in the following form: `Sun Sep 16 01:03:52 1973` Note that you can extract parts of this fixed-format string with @substr().

@year(e) — Return the year. Valid years begin with 1970. The last legal year is system dependent.

@month(e) — Return the month, encoded as 1 (January) to 12 (December).

@day(e) — Return the day of the month, encoded as 1 to 31.

@hour(e) — Return the number of hours since midnight, encoded as 0 to 23.

@minute(e) — Return the number of minutes since the last full hour, encoded as 0 to 59.

@second(e) — Return the number of seconds since the last full minute, encoded as 0 to 59.

Spreadsheet Update

Re-evaluation of spreadsheet expressions is done by row or by column depending on the selected calculation order. Evaluation is repeated up to iterations times for each update if necessary, so forward references usually work as expected. See set above. If stability is not reached after ten iterations, a warning is printed. This is usually due to a long series of forward references, or to unstable cyclic references (for example, set A0's expression to ''A0+1'').

Bugs

Top-to-bottom, left-to-right evaluation of expressions is silly. A proper following of the dependency graph with (perhaps) recourse to relaxation should be implemented.

Supports at most 200 rows and 40 columns.

Editing is crude. All you can do is backspace over and retype text to be altered. There is no easy way to switch a leftstring to a rightstring or vice versa. Of course, you can always write the spreadsheet to a file with Put, edit it by calling an editor on the file with ‘‘!’’, and read it back with Get - if you are comfortable editing spreadsheet files.

Only one previous value is saved from any call of @ext(). If it is used more than once in a spreadsheet and external functions are enabled and later disabled, the last returned value pops up in several places.

On some systems, if the cell cursor is in column 0 with topline enabled (so the current cell is highlighted), or if any cell in column 0 is highlighted, the corresponding row number gets displayed and then blanked during a screen refresh. This looks like a bug in curses.

Many commands give no indication (a message or beep) if they have null effect. Some should give confirmation of their action, but they don’t.

Associated files

`a1616.t1, environ, sc.doc, sc.exec, sc.man, tutorial.sc` (source isn’t included because it is too large to compile on a standard Applix 1616 - if you want a copy, ask).

Distribution

Applix 1616 Shareware Disk # 15 /sc

Author

Release 3.0, Ported to Applix 1616 by Andrew Morton.

scan - list keyboard scan codes - Andrew Morton - SW#9

scan

Description

A little demo program that intercepts the keyboard scan code vector, and places the scancode in a private buffer. Read by calls to ***getscancode***. Better than **kv**.

The 1616 keyboard is interrupt driver: every time a key is pressed (or released), a hardware interrupt is generated. The code produced by the keyboard is passed by an EPROM routine to a routine at the address given by ***set_kvec()***. The default routine in EPROM converts the scan codes to ASCII, and places the ASCII character in the keyboard buffer ready for other use.

You can replace the EPROM interrupt service routine (ISR) with your own code, using ***set_kvec()*** to point to your code. This is done in `scan.c`. Note that you can still cal the existing EPROM routine after your own routine, as done in `scan.c`

Use Esc to exit.

Associated files

`scan.xrel, scan.c, scancode.doc, makefile`

Distribution

Applix 1616 Shareware Disk # 9 /scancodes

Author

Andrew Morton

sdate - set time and date, using pop up window - Andrew McNamara - SW#6

sdate `[ -h]`

Description

Very cute little time and date set utility, that appears in a pop up window. After operation, it restores the display to its previous contents. Default operation is a 24 hour clock. Seems to work fine.

-h 12 hour clock option.

Use space bar or tab to move, backspace to reverse.

Quibbles

Time is not updated on pop up display during its operation. I can't see this as being a problem, but thought I needed some complaint other than merely about Andrew's vivid green colour scheme! Andrew tells me he only has a monochrome monitor, and has never seen it in colour.

Associated files

`sdate.xrel`, `sdate.c`

See also

dateset

Distribution

Applix 1616 Shareware Disk #6 /utility

Author

Andrew McNamara

sdate - set time and date - Matthew Gardener - SW#15

sdate

Description

Time and date set utility. Requests time and date in a human (not USA) fashion. Converts to system format, sets system date and time. Tests briefly for numerics.

Quibbles

There are too many of these routines!

Associated files

`sdate.xrel`, `sdate.s`

See also

dateset, **sdate**

Distribution

Applix 1616 Shareware Disk #15

Author

Matthew Gardener

sega - EGA style video drivers - Conal Walsh - SW#6

sega.mrd

Description

Early version of EGA 640 by 350 video drivers, and Microbee compatible video drivers. See tutorials in µPeripheral #10 and on.

Examples

Quibbles

Associated files

`/ega, /mbv, asmv.shell, info.video, syscalls.newlib, video.global`

See also

Distribution

Applix Shareware Disk #6 /video /video/ega, /video/mbv

Author

Conal Walsh

setvol - change name of disk volume - Michael Johnson - SW#15

setvol `name` [`device`]

Description

Allows you to easily change the volume name on a disk. Excellent documentation in Michael's usual style.

Associated files

`setvol.xrel`, `setvol.c`, `setvol.h`, `setvol.doc`

Distribution

Applix 1616 Shareware Disk # 15 /johnson

Author

Michael Johnson, 11/12 Kokoda Street, Wagga Wagga. 15 Aug 1989.

terminal - replacement for inbuilt terminal ??? - SW#14

terminal `port` [`ver`] [`half`] [`inLF`] [`outLF`] [`print`] [`echo`] [[`-`]`logfile`]

Description

This terminal program is designed as an update to replace the 1616/OS inbuilt terminal emulation. The optional arguments can be used in any order. The meaning of the arguments are as follows:

port	Serial port `sa:` or `sb:`
ver	Verbose mode on.
half	Half duplex operation on.
inLF	add line feeds to incoming carriage returns.
outLF	add line feeds to outgoing carriage returns.
print	echo all data to Centronics port.
echo	echo all received data back through the output port.
logfile	file to which all output is directed.
-	appends to logfile whose name is given.

Associated files

`terminal.doc`

Distribution

Applix 1616 Shareware Disk # 14

Author

?

testmrd - test MRD prior to installing, by Michael Johnson - SW#5

author:- MICHAEL JOHNSON, WAGGA WAGGA
program name:- TESTMRD.XREL
program source code:- TESTMRD.S
date:- 13-9-88
version :- 1.0
document issue :- 1

INTRODUCTION

This program is designed to test an MRD before it is installed into the MRDRIVERS file on you disc. The program as supplied does perform it's functions correctly. However, it may cause some disturbances to your system when you run it. This is due to some unknown causes which I hope to find eventually. The program will always send a level 0 restart command to the MRD and report the returned value. You will need to inform the TESTMRD function of any other tests you wish to perform. If the MRD is enabled and you ask for an execentry command to be sent the program will also permitt you to test the 1616o/s command level of the MRD IF it exists.

USING THE TESTMRD FUNCTION

The command is typed in at 1616o/s level. The mrd MUST be in .XREL format AND must have an .XREL extension. If the enable, disable, doit or stopit commands need an argument then the program will permit you to send the argument to the mrd with these commands. Each command requires it's own argument.

The command line structure :-

```
testmrd mrd_name.xrel [-e (argument)] [-d (argument)]  [-x
(argument)] [-s (argument)]  [-t (number)]
```

Where :-

The parts in square brackets '[]' are optional,
The parts in standard brackets '()' are mandatory if the option is selected
DO NOT include the brackets.

Each indicator and argument must be separated by at least 1 space.

indicator	-e	will supply an argument for enable mrd call
indicator	-d	will supply an argument for disable mrd call
indicator	-x	will supply an argument for doit mrd call
indicator	-s	will supply an argument for stopit mrd call
indicator	-t	is used to state which tests are to be performed.

1 will test level 1 restart
2 will test level 2 restart
3 will test get name command
4 will test get version command
5 will test enable command
6 will test disable command
7 will test doit command
8 will test stopit command
9 will test execentry command
.255 will test all of the above commands

If you supply any other value to the test indicator it will print the following error message :-

number too large

If the program has trouble working out your arguments then it will print a usage message similiar to this one

USAGE:- testmrd mrd_name.xrel [-e (argument)] [-d (argument)] [-x (argument)] [-s (argument)] [-t (number)]

-e will supply an argument for enable mrd call
-d will supply an argument for disable mrd call
-x will supply an argument for doit mrd call
-s will supply an argument for stopit mrd call

-t(number) where 0 < (number) < 10 OR (number) = .255 (FF)this number is the mrd command number .255 (FF) will execute all mrd commands

error at argument :- 9

If you find this documentation is too short or not informative enough then feel free to write to me and state what you think should be added, changed or improved.

KIND REGARDS

MICHAEL JOHNSON

(069) 255255

teststt14 - password protection - Ole Nielson - SW#16

teststr14 `password`

Description

A little password security program. Cleans the screen and doesn't let you use it until you enter the correct password.

When invoked, the message
```
Machine is in use
Do not touch
```
appears on screen, and rearranges itself continually, which is a cute touch.

Associated files

`teststr14.c`

Distribution

Applix 1616 Shareware Disk # 16

Author

Ole Nielson, 5 Aug 1989.

toupper - filter lower case to upper - Matthew Gardener - SW#15

toupper

Description

A filter to convert lower case to upper case. Invoke it via the pipe (|) routine.

Associated files

`toupper.s`

Distribution

Applix 1616 Shareware Disk # 15 /gardener

Author

Matthew Gardener, PO Box 8021, Palmerston North, NZ.

trace - single steps a program - Mike Gregory - SW#3

trace `address`

Description

Traces the execution of a program, by forcing it to single step.

x exits from **trace**

single step on any other key

Examples

I haven’t tested it - has anyone?

Quibbles

Associated files

`trace.s`

See also

except

Distribution

Applix 1616 Shareware Disk #3 /utilities

Author

Mike Gregory (in *ETI July 1988)*

tree - lists all directories and files - Matthew Geier - SW#3

tree [**-q**] [/vol]
tree [**-q**] [device name]

Description

Tree lists all the directories and files on a block device, such as a disk. If there is no argument, it uses the currently active device. It is intended to be able to find a file, including its full path, by piping through **grep** or a similar search utility.

-q Quiet mode, lists file names only.

The listing is done one file per line. Directory and backup bits are reported.

Quibbles

Has trouble returning with really deep directory trees (sometimes fails to print the whole path).
Quiet mode sometimes forgets that it is supposed to be quiet.

Associated files

tree.xrel, tree.c, filesys.h

See also

whereis (Andrew wrote one recently)

Distribution

Applix 1616 Shareware Disk #3 /utilities/tree (early version, may not include **-q** option).

Author

Matthew Geier

ymodem - ymodem file transfer - Colin McCormack - SW#8

ymodem `sa: [sb:]`

Description

Yet another modem program, this one providing `ymodem` protocols (don't ask me the difference).

Possible options:

c crc error checking (otherwise uses checksum).

s serial port `sa:` or `sb:`

v verbose mode invoked.

Examples

Quibbles

No `.exec` files with this, couldn't even be sure who wrote it.

Associated files

`crcy.c, log.c, main.c, modem.c, rxy.c, ry.c, txy.c, ymodem.h`

See also

modem, rb, **sb**, sealink(?)

Distribution

Applix 1616 Shareware disk #8 /ymodem

Author

Colin McCormack

Z80asc - Z80 memory map dump - Sid Young - SW#6

Z80asc

Description

Dumps the contents of the Z80 memory, in ASCII.

Quibbles

Associated files

`z80hex.exec`, `z80asc.s`

See also

z80hex

Distribution

Applix 1616 Shareware Disk #6 /utility/sid

Author

Sid Young

Z80hex - Z80 memory map dump - Sid Young - SW#6

Z80hex

Description

Dumps the contents of the Z80 memory, in hexadecimal.

Quibbles

Bombs on my system (but strangely **z80asc** works?)

Associated files

`z80hex.exec`, `z80hex.s`

See also

z80asc

Distribution

Applix 1616 Shareware Disk #6 /utility/sid

Author

Sid Young

zrdos - transfer files between zrdos and 1616 - Joe Moschini - SW#11

atoz `filename`
ztoa
zd

Description

Set of programs to allow you to transfer files between the 1616 file system, and a Zrdos CP/M file system.

First transfer `zrdos.ram` to `/rd`. Absolutely essential.

`zd` displays Zrdos directory, with its bit map.

`atoz filename` transfers 1616 files to Zrdos files. It accepts wildcards.

`ztoa` transfers Zrdos files to 1616. Press the Space bar for the next file. Press Enter to actually transfer the displayed file. Press Esc to exit program.

Associated files

`atoz.c, ztoa.c, zd.c, atoc.xrel, ztoa.xrel, zrdos.ram`

Distribution

Applix 1616 Shareware Disk # 11 /zrdos

Author

Joe Moschini, 33 Wade Street, Embleton, Perth, W.A. 6062, who got tired of moving them the hard way.

Summary

arc [**-**]{amufdxeplvtc} [biswn] [g*password*] *archive* [*filename ...*]

asciicalc

basic

bigbuf [bufsize] [satx] [sarx] [sbtx] [sbrx] [cent] [kb]

bmedit bmfile

c1616 helpfile [swapfile]

calendar.bas

cdev [device:]

dateset.bas

dateset

demonstration

dhryreg
dhrynoreg

dial phone

disassemble file.xrel [char]
disassemble m address1 address2 [char]

diskio

du2

easy_write

ep232

except

except.mrd

factorial [-v] number

ftree [-s filename] [-d] [-f] [/dev]

getty [sb:] [-u*id*] [-m*alrwx*] [-c*barfile*]

hs name *or* number

indent [input-file [output-file]] [-bad | -nbad] [-bap | -nbap] [-bbb | -nbbb] [-bc | -nbc] [-bl] [-br] [-cn] [-cdn] [-cdb | -ncdb] [-ce | -nce] [-cin] [-clin] [-dn] [-din] [-fc1 | -nfc1] [-in] [-ip | -nip] [-ln] [-lcn] [-lp | -nlp] [-pcs | -npcs] [-npro] [-psl | -npsl] [-sc | -nsc] [-sob | -nsob] [-st] [-troff] [-v | -nv]

kmem

kv

lfk `fkey_def_file`

lisp

locate.mrd

makeega

mem

modem `sa:` [`sb:`]

modem32

more `[ files ... ]`

nswp

pfile.bas

prog?

ps3

quick.mrd

quindex

rb [`-7buv`] [`-sA`] [`-sB`]
rb [`-1bcuv`] `filename`

readmrd

sb [`-7dfkquv`] [`-sA`] [`-sB`] `filename ...`

sc `[ -c ] [ -m ] [ -n ] [ -r ] [ -x ] [ file ]`

scan

sdate `[ -h]`

sdate

sega.mrd

setvol `name` [`device`]

terminal `port` [`ver`] [`half`] [`inLF`] [`outLF`] [`print`] [`echo`] [[`-`]`logfile`]

testmrd

teststr14 `password`

toupper

trace `address`

tree [**-q**] [`/vol`]
tree [**-q**] [`device name`]

ymodem `sa: [sb:]`

Z80asc

Z80hex

atoz `filename`
ztoa
zd

Index

bigbuf buffer size, 5
buffer size bigbuf, 5

Table of Contents

A Note to Authors 1

arc - compress and archive files - Andrew Morton 2

asciicalc - ascii string evaluation tutorial - Sid Young - SW#6 3

basic - Tiny basic - Andrew Morton - SW#14 4

bigbuf - set buffer size for character devices - Andrew Morton - SW#14 5

bmedit - bit map editor - Andrew McNamara - SW#15 6

c1616 - help system - Conal Walsh - SW#13 7

calendar - prints whole year calendar - Paul Cahill - SW#4 9

cdev - dump character device tables - Andrew McNamara - SW#15 10

dateset - set system clock - David Fowler - SW#4 11

dateset - at power up, by Michael Johnson - SW#5 12

debug - breakpoint debugger - Gerhard Baumann - SW#14 13

demonstration - of exception handler by Michael Johnson - SW#5 14

dhrystone - c compiler benchmark - Andrew Morton - SW#9 15

dial - simple phone autodialler - Matthew Geier - SW#15 16

disassemble - disassemble memory or file - Gerhard Baumann - SW#14 17

diskio - disk input output calls tutorial - Sid Young - SW#6 18

du2 - select and view disk blocks - Sid Young - SW#6 19

Easy_write - group and modify programs - Michael Johnson - SW#15 20

ep232 - eprom burner software - Joseph Moschini - SW#3 24

except - exception handler for bug tracing - Michael Johnson - SW#3, SW#5 25

Except MRD, by Michael Johnson - SW#3, SW#5 26

factorial - evaluates factorials - Gerhard Baumann - SW#15 41

ftree - find files and display directory trees - Michael Johnson - SW#15 42

getty - restricted shell via serial port - Andrew McNamara - SW#15 45

hs - Help for system calls - Conal Walsh - SW#13 46

indent - C programs formatter- Andrew Morton - SW#10 47

kmem - C storage allocator - Andrew Morton - SW#6 52

kv - prints keyboard scan codes - Andrew Morton - SW#6 53

lfk - define function keys - Cameron Hutchison - SW#15 54

lisp - ai language - Andrew Morton - SW#8 55

locate - MRD to find data in memory, by Michael Johnson - SW#5 56

makeega - run Hercules monitor as EGA - Lindsay Washusen - SW#15 64

mem - memory free report - Andrew McNamara - SW#15 65

modem - Xmodem file transfer - Mark Harvey - SW#1, SW#2 66

modem32 - menu driven file transfer program - Sid Young - SW#6 67

more - formats text files for viewing - Matthew Gardner - SW#15 68

nswp32 - bulk file copy, delete, view utility - Sid Young - SW#6 70

pfile - menu driven flat file data base - Paul Cahill - SW#4 72

prog? - Assembler tutorial examples - Kathy Morton - SW#1 73

ps3 - process status nice value - Matthew Geier or Andrew McNamara - SW#15 74

quick - mrd to speed up processing - Andrew Morton - SW#9 75

quindex - quick index - Michael Johnson - SW#5 76

rb - receive yam or ymodem files - Colin McCormack - SW#3 78

readmrd - reports on MRDs - Michael Johnson - SW#3, SW#5 79

sb - send yam or ymodem files - Colin McCormack - SW#6 81

sc - spreadsheet calculator - Andrew Morton - SW#15 82

scan - list keyboard scan codes - Andrew Morton - SW#9 99

sdate - set time and date, using pop up window - Andrew McNamara - SW#6 ... 100

sdate - set time and date - Matthew Gardener - SW#15 101

sega - EGA style video drivers - Conal Walsh - SW#6 102

setvol - change name of disk volume - Michael Johnson - SW#15 103

terminal - replacement for inbuilt terminal ??? - SW#14 104

testmrd - test MRD prior to installing, by Michael Johnson - SW#5 105

testst14 - password protection - Ole Nielson - SW#16 107

toupper - filter lower case to upper - Matthew Gardener - SW#15 108

trace - single steps a program - Mike Gregory - SW#3 109

tree - lists all directories and files - Matthew Geier - SW#3 110

ymodem - ymodem file transfer - Colin McCormack - SW#8 111

Z80asc - Z80 memory map dump - Sid Young - SW#6 112

Z80hex - Z80 memory map dump - Sid Young - SW#6 113

zrdos - transfer files between zrdos and 1616 - Joe Moschini - SW#11 114

Summary 115