MT XINU

UNIX™ SUPPORT FROM BERKELEY

4.3 BSD with NFS

User Contributed Software

UCS

MTXINUNIXTMTXINUNIXTMTXINUNIXTMTXINUNIX

MT XINU™

UNIX™ SUPPORT FROM BERKELEY

User Contributed Software

The subtree /usr/src/new contains programs contributed by the user community. The following software is included:

Directory	Description	Contributor(s)
B	B progamming language & environment	CWI
X	X Window system	M.I.T.
ansi	ANSI and VMS standard tape handler	Tom Quarles, Berkeley
apl	APL system	Purdue
bib	bibliography system	Arizona
courier	remote procedure call package	Eric Cooper, Berkeley
cpm	CP/M floppy access package	Helge Skrivervik
dipress	Xerox Interpress Tools	Xerox
dsh	distributed shell	Dave Presotto, Berkeley
emacs	Gnumacs	Richard Stallman
enet	Packet filter	Jeff Mogul, Stanford
help	Help system	John Kunze, Berkeley
hyper	Hyperchannel support tools	Steve Glaser, Tektronix
icon	ICON system	Arizona
jove	Emacs editor	Jon Payne
kermit	File transfer protocol	Columbia University
mh	MH mail system	Rand Corporation
mkmf	Makefile generator	Peter Nicklin, Berkeley
mmdf	MMDF mail system	Dr. Dave Farber, Delaware
news	"readnews" bulletin board system	Matt Glickman, Berkeley
notes	notes files bulletin board system	Illinois
np100	Utilities for Interlan NP100	MICOM-Interlan
patch	apply diffs to originals	Larry Wall, SDC
pathalias	uucp router	Peter Honeyman, Princeton
rcs	revision control system	Walter Tichy, Purdue
rn	readnews front end	Larry Wall
spms	software project management system	Peter Nicklin, Berkeley
sumacc	MacIntosh cross development system	William Croft, Stanford
sunrpc	Remote procedure call package	Sun Microsystems
tac	reverse a file by segments	Jay Lepreau, Utah
tools	miscellaneous tools	John Kunze, Berkeley
umodem	File transfer protocol	Lauren Weinstein
xns	XNS/Courier user code	J.Q. Johnson, Cornell

The individuals responsible for the software should be identified in the accompanying 4.3BSD documents which describe the user contributed software. All software included here has been written by outside parties; we gratefully acknowledge their contributions.

Consult "Installing and Operating 4.3BSD on the VAX" (SMM:1) for instructions on how to extract the user contributed software. The organization of the software is such that a single make command will compile and/or install most of it. Some of the software requires customization before it can be built and installed. The software requiring customization is listed in the top level Makefile in OPTDIR. To compile everything else, simply type

```
make
```

Once this is done, to install the default software in the /usr/new area of the file system type

```
make installall
```

to install only the subset of software installed on the distribution tape type

```
make install
```

Most subdirectories have README files and individual Makefiles. If you want only some of the software contained here go to the appropriate directories and use the "make" and "make install" commands to compile and install the desired system. As distributed, all the default software is set up to be installed in ${DESTDIR}/usr/new, where DESTDIR is a make macro that can be supplied on the command line. Consult each directory's README file for the information needed to change this.

The software included here is in use at Berkeley, or other sites running 4.3BSD (or an earlier derivative). Please remember that this is contributed software and, as such, we do not "support" it in the same manner as that software which is part of the standard distribution. Most subsystems have either a README file or doc directory that should be consulted to find an interested party to which bugs and/or bug fixes should be sent. In certain cases these people are located at Berkeley; this does not imply they are part of the Computer Systems Research Group, please contact them as individuals.

Mike Karels
Kirk Mckusick
Jim Bloom
Miriam Amos
Kevin Dunlap

USER CONTRIBUTED SOFTWARE (UCS)

4.3 Berkeley Software Distribution
Virtual VAX–11 Version

April, 1986

Computer Systems Research Group
Computer Science Division
Department of Electrical Engineering and Computer Science
University of California
Berkeley, California 94720

USER CONTRIBUTED SOFTWARE (UCS)

4.3 Berkeley Software Distribution
Virtual VAX–11 Version

April, 1986

Computer Systems Research Group
Computer Science Division
Department of Electrical Engineering and Computer Science
University of California
Berkeley, California 94720

INDIVIDUAL MANUAL PAGES
(UCS)

NAME
courier – Courier remote procedure call compiler

SYNOPSIS
courier [**–x**] specification

DESCRIPTION
Courier is a compiler for the Mesa-like specification language associated with the Courier remote procedure call protocol.

FILES

Program.cr	Courier specification file for *Program*

The following files are generated by courier from the above:

Program.h	definitions and typedefs
Program_stubs.c	mappings between C and Courier
Program_server.c	server routines
Program_client.c	client routines

SEE ALSO
"Writing Distributed Programs with Courier" by Eric C. Cooper.
"Courier: The Remote Procedure Call Protocol," Xerox System Integration Standard 038112, December 1981.

NAME

cpm – read and write CP/M® floppy disks

SYNOPSIS

cpm [options] [filename]

DESCRIPTION

Cpm reads and writes files with an internal structure like a CP/M file system. By default *cpm* assumes that the specified file has the parameters of a standard IBM format single sided single density 8" CP/M floppy disk, i.e., 2002 records containing 128 bytes each, of which 52 are reserved for system use and 16 (2 blocks) are used by the directory (maximum 64 directory entries). These parameters may be changed by specifying the appropriate flags (see below). Thus, various double density formats may also be read and written, provided that the hardware can handle the actual format.

The specified file may be a floppy disk drive (e.g., /dev/floppy on an 11/780 or /dev/rrx?b if rx02 drives are available on your system), or a standard UNIX file with the appropriate structure. Since it may be inconvenient (and slow) to access the device directly, in particular the console floppy on an 11/780, it is always a good idea to copy the contents of the diskette into a standard file using *dd*(1), e.g.,

```
dd if=/dev/floppy of=yourfile bs=128 count=2002
```

On most systems you have to be superuser to access the console floppy and to be able to write to rx02's.

Flags:

–d	display directory on standard output
–B	the files specified with the **c** or **C** flag contain binary code rather than plain text (default)
–c *name1 name2*	copy the CP/M file *name1* to the UNIX file *name2*
–C *name1 name2*	copy the UNIX file *name1* to the CP/M file *name2*
–p *name*	copy the specified CP/M file to standard output
–i	enter interactive mode (all the above flags are turned off)
–I	force initializtion of the specified CP/M file (e.g., delete all files)
–s*n*	skew factor (sector interleaving); default is 6
–b*n*	block size (in bytes); default is 1K bytes
–m*n*	max number of directory entries; default is 64
–l*n*	sector size (in bytes); default is 128
–r*n*	number of sectors per track; default is 26

If the **–i** flag is specified, the filename argument must always be present. If the specified file does not exist, a new file will be initialized. The **–C**, **–c** and **–p** flags are mutually exclusive.

The following commands are available in interactive mode:

ccopyin *unixfile cpmfile*	copy UNIX binary file to CP/M
ccopyout *cpmfile unixfile*	copy CP/M binary file to UNIX
copyin *unixfile cpmfile*	copy UNIX text file to CP/M
copyout *cpmfile unixfile*	copy CP/M text file to UNIX
del[ete] *filename*	a synonym for *erase*

dir[ectory] or **ls**	display directory
era[se] *filename*	delete the given file
hel[p]	print a short description of each command
log[out] or **exi**[t] or ^D	terminate, return to the shell
ren[ame] *file1 file2*	rename *file1* to *file2*
typ[e] *filename*	print CP/M file to console

The commands may be abbreviated as indicated by brackets. CP/M file names are automatically converted to upper case. The copy commands refuse to overwrite existing files.

If the CP/M floppy file becomes full during a file transfer from UNIX, the file is closed and the command terminated. The data already written to the CP/M file will be saved.

The *copyout* command assumes that CP/M text files have cr+lf as line terminators and removes carriage returns. *Copyin* adds a carriage return in front of each line-feed, and adds a ^Z to the end of the file. The binary copy commands provide for "raw" file copying, thus making it possible to copy code files to and from diskettes.

Interrupts are recognized in interactive mode, and will return you to the command level.

FILES

/dev/floppy
/dev/rrx?b
/usr/new/lib/cpm.hlp

SEE ALSO

dd(1), rx(4v)

BUGS

CP/M user numbers are ignored, files written to the CP/M floppy file will always have user number 0.

No testing has been done with double density floppies.

CP/M filename extensions containing more than 3 characters will quietly be truncated.

Wildcards are not supported.

The distinction between text and binary files is clumsy but necessary because CP/M uses CR/LF for line termination.

AUTHOR

Helge Skrivervik

NAME

dsh – distributed shell

SYNOPSIS

dsh [**–a**] [**–v**] [**–h** host] [**–n**] [**–i** copyto] [**–o** copyback] command

DESCRIPTION

Dsh selects a host and executes the specified *command* on it. If the command specifies a host using the **–h** option, that host is used. Otherwise the selection algorithm attempts to select the least loaded of the hosts. At the moment "least loaded" corresponds to the lowest load average. *Dsh* copies its standard input to the remote command, the standard output of the remote command to its standard output, and the standard error of the remote command to its standard error. Interrupt is propogated to the remote system. *Dsh* normally terminates when the remote command does.

The host is selected from a list of hosts. A default for this list exists on each system in a file called */usr/lib/dshrc.* This list can be overridden by an entry in a *.dshrc* file in the user's home directory. The format of the entry is:

```
hosts = [<weight>]<host>, ... , [<weight>]<host>
```

where <host> can be simply a host name or a binary tuple of the form:

```
(<host name>, <account name>)
```

where <weight> is a multiplier of the form:

```
<decimal number>*
```

The account used to run the command is by default the account of the user executing the dsh. This can be overridden using the second form of the host specification shown above. For example a user "mini" that wanted to execute commands on HOST1 as herself and on HOST2 as "mickey" would have a *.dshrc* file with the entry:

```
hosts = HOST1, (HOST2, mickey)
```

Of course "mickey" must have an entry in his .rhosts file to allow "mini" to use his account.

If "mini" wanted to wieght HOST1 so that it would be used even when its load average was twice that of HOST2 she would use:

```
hosts = 2.1*HOST1, (HOST2, mickey)
```

The directory created to run the command in on the remote machine is normally in the account's login directory. This can be overridden by another entry in the *.dshrc* file of the format:

```
dir = <directory name>
```

If the *–v* option is specified *dsh* reports the name of the machine the command is executed on.

Shell metacharacters which are not quoted are interpreted on local machine, while quoted metacharacters are interpreted on the remote machine. Thus the command

```
dsh cat remotefile >> localfile
```

appends the remote file *remotefile* to the localfile *localfile,* while

```
dsh cat remotefile ">>" otherremotefile
```

appends *remotefile* to *otherremotefile.*

The *–i* option is used to transfer the *copyto* file to the remote host before the command is executed. More than one *–i* option may be specified.

The *-o* option is used to transfer the *copyback* file back from the remote host after the command is executed. More than one *-o* option may be specified.

The *-a* option causes *dsh* to try to execute the command on as many hosts as it can.

Host names are given in the file */usr/lib/hosts.* Each host has one standard name (the first name given in the file), which is rather long and unambiguous, and optionally one or more nicknames.

FILES

~/.dshrc	the user's initialization file
/usr/lib/dshrc	the system defaults
/usr/ucb/dbid	the bidder (must exist on each machine)

SEE ALSO

rlogin(1x), rpasswd(1x), rsh(1x)

BUGS

If you are using *csh*(1) and put a *dsh*(1x) in the background without redirecting its input away from the terminal, it will block even if no reads are posted by the remote command. If no input is desired you should redirect the input of *dsh* to /dev/null using the **-n** option.

You cannot run an interactive command (like *rogue*(6) or *vi (6));* use *rlogin*(1x).

Stop signals stop the local *dsh* process only; this is arguably wrong, but currently hard to fix for reasons too complicated to explain here.

NAME

help – an easy way to find and use information

SYNOPSIS

help [**–d** dirlist] [**–m** key] [**–p** prompt] [**–i**] [**–n**] [**–q**] [topic [subtopic [subsubtopic [...]]]]

DESCRIPTION

The primary purpose of *help* is to provide easy access to on-line documentation. In response to the command **help**, the user is placed in an interactive setting and presented with a list of topics and a set of instructions to perform on them. To bypass the interactive part and just display what is known about a topic, enter the topic name on the command line after **help**.

A topic is displayed by typing its name or a unique abbreviation, and topics may be requested and listed with numbers. Topics may be saved in a file or printed on the lineprinter. When you request a topic not on its list, *help* can search a set of indexes into its own files, the Unix Programmer's Manual, and various optional off-line sources. References found in this way can be displayed if material is available on-line. All topic requests, including ones which yield no useful information, are automatically recorded for later analysis by system maintainers. Any user may effectively add topics to **help**'s knowledge base from that user's point of view by suitably defining the environment variable HELPPOOL. Thus each user can have a private cache of topic files accessible with *help*.

In general the purpose of *help* is to provide a way to move around easily in a set of hierarchical databases, namely, one or more Unix file system subtrees. Although an inherent disadvantage of such databases is that data may be hidden within the hierarchies, the program overcomes this to some extent with its indexing feature. *Help* can be made to serve in special purpose applications without programmer intervention.

The following technical description may be of little interest to casual users.

The default knowledge base consists of the files and directories in the subtree /usr/lib/help/cat. If the environment variable HELPPOOL contains a list of directories (separated by spaces or colons), their contents are merged with the default list to form the knowledge base. The entire directory pool may be overridden by giving a directory list in *dirlist* after the **–d** option.

The interactive user prompt is by default the tail of argument zero from *exec*(3), usually "help", followed by a list of directories leading to the current subtopic directory. A different prompt may be specified by making a link with a name of your choice to /usr/ucb/help or by specifying a name after the **–p** option. A special prompt is used in help-index mode, which is entered when the user initiates an index search on a given keyword. The string "-index" and the keyword are added to the prompt in this case.

Help may be invoked with any number of *topic*, *subtopic*, *subsubtopic*, etc., arguments, which may be abbreviated. Starting at the top directory level *help* tries to change to each directory named by successive arguments, interactively resolving non-unique abbreviations along the way. When a name corresponds to a file, the file is displayed and, unless **–i** was specified, the program exits. If the standard output is not a terminal, *help* refuses to be interactive unless the **–i** option is given. If the last valid name is a directory, or no topics were requested, *help* lists the topics at the current level and prompts for user input.

Maintainers of *help*-style knowledge bases can use the **–m** option to perform various custodial functions using shell scripts. Permission to use this option is restricted to those users who have write permission for one of the directories in the list specified after the **–d** option or in HELPPOOL. If such a directory exists, it (the first one) becomes the value of the variable $subtree, and the local maintenance configuration is sourced from $subtree/../maint/config before most functions are performed. The function name, given as *key* after **–m** , may be followed by other arguments (but no topic abbreviations are recognized here), in which case the script $subtree/../maint/do.*key* is invoked. In the absence of *key*, the word "default" is used, usually causing the valid function choices to be listed.

A simple macro package called **-mayday** is used by authors to format *help* topic files. Its purpose is to standardize somewhat the display format for topic files, create a hook for the index generating script, and guarantee page headers and footers of a fixed length so that *help* will not strip out too much or too little of a topic file when displaying it without pagination. The only macro call required is the initializing macro, .TI, followed by a filename and an optional date on the same line, and by a keyword-rich title (to be useful for the index) on the next line. A handful of other macros are provided for compatibility with other packages, though they are usually unnecessary.

Authors of *help* topics may make topic files into shell scripts by entering a # or : at the beginning of the first line. This causes help to run the script when that topic is requested, allowing programs to be demonstrated, questions asked, etc. There are maintenance scripts which will convert a script with embedded *nroff* source text into a script with embedded formatted text, provided the lines to be formatted appear between the special lines "##nroff" and "##".

Similarly, authors may use programs as topic files, the source being written in an aribtrary programming language.

The **-n** option causes topics to be listed and accessible by numbers (and still accessible by unique abbreviations). The **-q** option suppresses the instruction line appearing before each prompt, including the prompt printed by the **-d** option to *more*(1), through which long output is piped after multiple blank lines in a row are reduced to one. Both of these features are available as instructions from within *help*.

The internal instructions are described below.

% or $
: Quit from help and return to the shell. Control-d works also.

topic
: Display *topic* on the terminal. *Topic* may be the shortest unique name abbreviating a topic at the current level. *Help* prompts for more characters if a non-unique name is given, and asks to look in the index if the name abbreviates no topic. If a name is given as = in any context other than index mode, the current topic is used, where the current topic is defined to be the one most recently accessed. In index mode, a name of = has no special significance, and if *topic* is a unique abbreviation for a reference, that reference is displayed.

topic +
: Enter index mode and see what more is known about *topic* by looking in the indexes. A missing topic is taken to be =.

topic > file
: Save *topic* in *file*. A missing topic is taken to be =, and a missing file is taken to be "helpsave". If *file* exists the topic file is appended to it.

topic >& file
: Save *topic* in *file*, preserving headers and footers. Normally, topics are stored as *nroff*'d text files, the headers and footers of which are suppressed when displayed on a terminal. They are preserved in *file* with this instruction.

topic | lpr
: Print *topic* on the lineprinter, preserving headers and footers. The *lpr* string may be replaced with another program name (such as *ipr* or *vpr*) followed by one optional argument. A missing *topic* is taken to be the current topic, and a missing *lpr* is taken to be "lpr".

?
: List the internal instructions and describe them briefly.

.
: List topics at the current level, indicating the current topic, if any, with an =. In index mode, list references for the current subject.

..
: Back up to and list the next higher level of topics. From index mode, this means leave index mode and continue at the previous topic level.

/ Back up to and list the top level of topics. From index mode, this means leave index mode and continue at the top level of topics.

< Send comments or other input to the maintainer of *help* via electronic mail.

!command
Do a Unix *command* and then return to help. All occurrences of = inside *command* will be replaced by the current topic, if any.

* flag **on**/**off**
Set *flag* **on** or **off** to adjust the behavior of the program. A missing value for *flag* means invert its current value, and * by itself means display the current flags, their settings, and what they signify. There are currently two flags, **n** and **q**, which control the same things as the **-n** and **-q** options to *help*.

FILES

/usr/lib/help/cat	root of system *help* files
/usr/lib/help/src	nroff sources for system *help* files
/usr/lib/help/log	log of user requests; can be removed
/usr/lib/help/maint	maintenance scripts
/usr/lib/help/cat/generalgeneral	introduction to *help*
/usr/lib/help/cat/index_*	used to locate further references
/usr/lib/tmac/tmac.ayday	macro package for *help* files

SEE ALSO

environ(7), exec(3), learn(1), lpr(1), more(1), nroff(1)

AUTHOR

John Kunze

BUGS

Pathnames inside topic names are not recognized.

Help is really just a weak, friendly shell. Strengthening it might be more painful and less useful than civilizing the shell.

NAME

hyroute – set the hyperchannel routine tables

SYNOPSIS

hyroute [**-s**] [**-p**] [**-c**] [**-l**] [**-d**] [*file*]

DESCRIPTION

Hyroute manipulates the Hyperchannel routing information.

With the **-s** option, it reads *file* and sets the system's database according to the information in the file (see below). If no input file is given, or if the argument '–' is encountered, *hyroute* reads from standard input.

The **-c** option causes hyroute to compare the system's current information to that contained in *file*.

The **-p** option causes a digested version of *file* to be printed.

The -d option causes a "dump" of the system's table (used for debugging routing code).

FILE FORMAT

The input file is free format. Comment lines start with a '*' in column one. Statements end with a semicolon.

direct *host dest control access ;*

Describes a host that can be directly reached from this adapter. *Host* is a host name as listed /usr/lib/hosts, *dest*, *control*, and *access* are hexadecimal numbers. The data will be send to hyperchannel address *dest* using a control value of *control* and an access code of *access* (see adapter manuals for details).

The specified remote adapter and the local adapter must both be connected to one or more common trunks or connected to trunks that are connected with with link adapters.

gateway *host gate1 gate2 gate3 ... ;*

Describes a host that must be reached indirectly through any one of the hosts indicated by *gate*n. The hosts listed are not gateways in the formal sense (they don't run the internet gateway protocols), but are hosts on the hyperchannel can "bridge" between subsections of the hyperchannel network.

A sample file follows:

```
* comment
direct azure      6100 0 0;
direct bronze     6101 0 0;
direct cyber      2100 1100 0;
direct dadcad     6102 0 0;
direct tekcad     2400 1100 0;
direct tekcrd     2201 1100 0;
direct tekid      2500 1100 0;
direct teklabs    2200 1100 0;
gateway iddic     tekcrd teklabs cyber tekcad tekid;
gateway iddme     tekcrd teklabs cyber tekcad tekid;
gateway metals    tekcrd teklabs cyber;
```

SEE ALSO

hy(4)

FILES

/dev/hy character special file to get to the interface (only has an ioctl entry)

BUGS

Probably.

AUTHOR

Steve Glaser, Tektronix Inc.

NAME

jot – print sequential or random data

SYNOPSIS

jot [options] [reps [begin [end [s]]]]

DESCRIPTION

Jot is used to print out increasing, decreasing, random, or redundant data, usually numbers, one per line. The *options* are understood as follows.

–r Generate random data instead of sequential data, the default.

–b word
Just print *word* repetitively.

–w word
Print *word* with the generated data appended to it. Octal, hexadecimal, exponential, ASCII, zero padded, and right-adjusted representations are possible by using the appropriate *printf*(3) conversion specification inside *word*, in which case the data are inserted rather than appended.

–c This is an abbreviation for **–w %c.**

–s string
Print data separated by *string*. Normally, newlines separate data.

–n Do not print the final newline normally appended to the output.

–p precision
Print only as many digits or characters of the data as indicated by the integer *precision*. In the absence of **–p**, the precision is the greater of the precisions of *begin* and *end*. The **–p** option is overridden by whatever appears in a *printf*(3) conversion following **–w**.

The last four arguments indicate, respectively, the number of data, the lower bound, the upper bound, and the step size or, for random data, the seed. While at least one of them must appear, any of the other three may be omitted, and will be considered as such if given as –. Any three of these arguments determines the fourth. If four are specified and the given and computed values of *reps* conflict, the lower value is used. If fewer than three are specified, defaults are assigned left to right, except for *s*, which assumes its default unless both *begin* and *end* are given.

Defaults for the four arguments are, respectively, 100, 1, 100, and 1, except that when random data are requested, *s* defaults to a seed depending upon the time of day. *Reps* is expected to be an unsigned integer, and if given as zero is taken to be infinite. *Begin* and *end* may be given as real numbers or as characters representing the corresponding value in ASCII. The last argument must be a real number.

Random numbers are obtained through *random*(3). The name *jot* derives in part from *iota*, a function in APL.

EXAMPLES

The command

```
jot  21  -1  1.00
```

prints 21 evenly spaced numbers increasing from –1 to 1. The ASCII character set is generated with

```
jot  -c  128  0
```

and the strings xaa through xaz with

```
jot -w xa%c 26 a
```

while 20 random 8-letter strings are produced with

```
jot -r -c 160 a z | rs -g 0 8
```

Infinitely many *yes*'s may be obtained through

```
jot -b yes 0
```

and thirty *ed*(1) substitution commands applying to lines 2, 7, 12, etc. is the result of

```
jot -w %ds/old/new/ 30 2 - 5
```

The stuttering sequence 9, 9, 8, 8, 7, etc. can be produced by suitable choice of precision and step size, as in

```
jot 0 9 - -.5
```

and a file containing exactly 1024 bytes is created with

```
jot -b x 512 > block
```

Finally, to set tabs four spaces apart starting from column 10 and ending in column 132, use

```
expand -`jot -s, - 10 132 4`
```

and to print all lines 80 characters or longer,

```
grep `jot -s "" -b . 80`
```

SEE ALSO

rs(1), ed(1), yes(1), printf(3), random(3), expand(1)

AUTHOR

John Kunze

BUGS

NAME

kermit – kermit file transfer

SYNOPSIS

kermit [option ...] [file ...]

DESCRIPTION

Kermit is a file transfer program that allows files to be moved between machines of many different operating systems and architectures. This man page describes version 4C of the program.

Arguments are optional. If *Kermit* is executed without arguments, it will enter command mode. Otherwise, *kermit* will read the arguments off the command line and interpret them.

The following notation is used in command descriptions:

fn	A Unix file specification, possibly containing either of the "wildcard" characters '*' or '?' ('*' matches all character strings, '?' matches any single character).
fn1	A Unix file specification which may not contain '*' or '?'.
rfn	A remote file specification in the remote system's own syntax, which may denote a single file or a group of files.
rfn1	A remote file specification which should denote only a single file.
n	A decimal number between 0 and 94.
c	A decimal number between 0 and 127 representing the value of an ASCII character.
cc	A decimal number between 0 and 31, or else exactly 127, representing the value of an ASCII control character.
[ ]	Any field in square braces is optional.
{*x,y,z*}	Alternatives are listed in curly braces.

Kermit command line options may specify either actions or settings. If *Kermit* is invoked with a command line that specifies no actions, then it will issue a prompt and begin interactive dialog. Action options specify either protocol transactions or terminal connection.

COMMAND LINE OPTIONS

–s *fn* Send the specified file or files. If *fn* contains wildcard (meta) characters, the Unix shell expands it into a list. If *fn* is '-' then *Kermit* sends from standard input, which must come from a file:

```
kermit -s - < foo.bar
```

or a parallel process:

```
ls -l | kermit -s -
```

You cannot use this mechanism to send terminal typein. If you want to send a file whose name is "-" you can precede it with a path name, as in

```
kermit -s ./-
```

–r Receive a file or files. Wait passively for files to arrive.

–k Receive (passively) a file or files, sending them to standard output. This option can be used in several ways:

```
kermit -k
```

Displays the incoming files on your screen; to be used only in "local mode" (see below).

```
kermit -k > fn1
```

Sends the incoming file or files to the named file, *fn1*. If more than one file arrives, all are concatenated together into the single file *fn1*.

```
kermit -k | command
```

Pipes the incoming data (single or multiple files) to the indicated command, as in

```
kermit -k | sort > sorted.stuff
```

-a *fn1* If you have specified a file transfer option, you may specify an alternate name for a single file with the **-a** option. For example,

```
kermit -s foo -a bar
```

sends the file foo telling the receiver that its name is bar. If more than one file arrives or is sent, only the first file is affected by the **-a** option:

```
kermit -ra baz
```

stores the first incoming file under the name baz.

-x Begin server operation. May be used in either local or remote mode.

Before proceeding, a few words about remote and local operation are necessary. *Kermit* is "local" if it is running on a PC or workstation that you are using directly, or if it is running on a multiuser system and transferring files over an external communication line — not your job's controlling terminal or console. *Kermit* is remote if it is running on a multiuser system and transferring files over its own controlling terminal's communication line, connected to your PC or workstation.

If you are running *Kermit* on a PC, it is in local mode by default, with the "back port" designated for file transfer and terminal connection. If you are running *Kermit* on a multiuser (timesharing) system, it is in remote mode unless you explicitly point it at an external line for file transfer or terminal connection. The following command sets *Kermit*'s "mode":

-l *dev* Line — Specify a terminal line to use for file transfer and terminal connection, as in

```
kermit -l /dev/ttyi5
```

When an external line is being used, you might also need some additional options for successful communication with the remote system:

-b *n* Baud — Specify the baud rate for the line given in the **-l** option, as in

```
kermit -l /dev/ttyi5 -b 9600
```

This option should always be included with the **-l** option, since the speed of an external line is not necessarily what you expect.

-p *x* Parity — **e**, **o**, **m**, **s**, **n** (even, odd, mark, space, or none). If parity is other than none, then the 8th-bit prefixing mechanism will be used for transferring 8-bit binary data, provided the opposite *Kermit* agrees. The default parity is none.

-t Specifies half duplex, line turnaround with XON as the handshake character.

The following commands may be used only with a *Kermit* which is local — either by default or else because the -l option has been specified.

-g *rfn* Actively request a remote server to send the named file or files; *rfn* is a file specification in the remote host's own syntax. If *fn* happens to contain any special shell characters, like '*', these must be quoted, as in

```
kermit -g x\*.\?
```

-f Send a 'finish' command to a remote server.

-c Establish a terminal connection over the specified or default communication line, before any protocol transaction takes place. Get back to the local system by typing the escape character (normally Control-Backslash) followed by the letter 'c'.

-n Like **-c**, but after a protocol transaction takes place; **-c** and **-n** may both be used in the same command. The use of **-n** and **-c** is illustrated below.

On a timesharing system, the **-l** and **-b** options will also have to be included with the **-r**, **-k**, or **-s** options if the other *Kermit* is on a remote system.

If *kermit* is in local mode, the screen (stdout) is continously updated to show the progress of the file transer. A dot is printed for every four data packets, other packets are shown by type (e.g. 'S' for Send-Init), 'T' is printed when there's a timeout, and '%' for each retransmission. In addition, you may type (to stdin) certain "interrupt" commands during file transfer:

Control-F:	Interrupt the current File, and go on to the next (if any).
Control-B:	Interrupt the entire Batch of files, terminate the transaction.
Control-R:	Resend the current packet
Control-A:	Display a status report for the current transaction.

These interrupt characters differ from the ones used in other *Kermit* implementations to avoid conflict with Unix shell interrupt characters. With System III and System V implementations of Unix, interrupt commands must be preceeded by the escape character (e.g. control-\).

Several other command-line options are provided:

-i Specifies that files should be sent or received exactly "as is" with no conversions. This option is necessary for transmitting binary files. It may also be used to slightly boost efficiency in Unix-to-Unix transfers of text files by eliminating CRLF/newline conversion.

-w Write-Protect — Avoid filename collisions for incoming files.

-q Quiet — Suppress screen update during file transfer, for instance to allow a file transfer to proceed in the background.

-d Debug — Record debugging information in the file debug.log in the current directory. Use this option if you believe the program is misbehaving, and show the resulting log to your local *Kermit* maintainer.

-h Help — Display a brief synopsis of the command line options.

The command line may contain no more than one protocol action option.

INTERACTIVE OPERATION

Kermit's interactive command prompt is "C-Kermit>". In response to this prompt, you may type any valid command. *Kermit* executes the command and then prompts you for another command. The process continues until you instruct the program to terminate.

Commands begin with a keyword, normally an English verb, such as "send". You may omit trailing characters from any keyword, so long as you specify sufficient characters to distinguish it from any other keyword valid in that field. Certain commonly-used keywords (such as "send", "receive", "connect") have special non-unique abbreviations ("s" for "send", "r" for "receive", "c" for "connect").

Certain characters have special functions in interactive commands:

?	Question mark, typed at any point in a command, will produce a message explaining what is possible or expected at that point. Depending on the context, the message may be a brief phrase, a menu of keywords, or a list of files.
ESC	(The Escape or Altmode key) — Request completion of the current keyword or filename, or insertion of a default value. The result will be a beep if the requested operation fails.
DEL	(The Delete or Rubout key) — Delete the previous character from the command. You may also use BS (Backspace, Control-H) for this function.
^W	(Control-W) — Erase the rightmost word from the command line.
^U	(Control-U) — Erase the entire command.
^R	(Control-R) — Redisplay the current command.
SP	(Space) — Delimits fields (keywords, filenames, numbers) within a command. HT (Horizontal Tab) may also be used for this purpose.
CR	(Carriage Return) — Enters the command for execution. **LF** (Linefeed) or **FF** (formfeed) may also be used for this purpose.
\	(Backslash) — Enter any of the above characters into the command, literally. To enter a backslash, type two backslashes in a row (\\). A single backslash immediately preceding a carriage return allows you to continue the command on the next line.

You may type the editing characters (DEL, ^W, etc) repeatedly, to delete all the way back to the prompt. No action will be performed until the command is entered by typing carriage return, linefeed, or formfeed. If you make any mistakes, you will receive an informative error message and a new prompt — make liberal use of '?' and ESC to feel your way through the commands. One important command is "help" — you should use it the first time you run *Kermit.*

Interactive *Kermit* accepts commands from files as well as from the keyboard. When you enter interactive mode, *Kermit* looks for the file .kermrc in your home or current directory (first it looks in the home directory, then in the current one) and executes any commands it finds there. These commands must be in interactive format, not Unix command-line format. A "take" command is also provided for use at any time during an interactive session. Command files may be nested to any reasonable depth.

Here is a brief list of *Kermit* interactive commands:

!	Execute a Unix shell command.
bye	Terminate and log out a remote *Kermit* server.
close	Close a log file.
connect	Establish a terminal connection to a remote system.
cwd	Change Working Directory.
dial	Dial a telephone number.
directory	Display a directory listing.
echo	Display arguments literally.
exit	Exit from the program, closing any open logs.
finish	Instruct a remote *Kermit* server to exit, but not log out.

get	Get files from a remote *Kermit* server.
help	Display a help message for a given command.
log	Open a log file — debugging, packet, session, transaction.
quit	Same as 'exit'.
receive	Passively wait for files to arrive.
remote	Issue file management commands to a remote *Kermit* server.
script	Execute a login script with a remote system.
send	Send files.
server	Begin server operation.
set	Set various parameters.
show	Display values of 'set' parameters.
space	Display current disk space usage.
statistics	Display statistics about most recent transaction.
take	Execute commands from a file.

The 'set' parameters are:

block-check	Level of packet error detection.
delay	How long to wait before sending first packet.
duplex	Specify which side echoes during 'connect'.
escape-character	Character to prefix "escape commands" during 'connect'.
file	Set various file parameters.
flow-control	Communication line full-duplex flow control.
handshake	Communication line half-duplex turnaround character.
line	Communication line device name.
modem-dialer	Type of modem-dialer on communication line.
parity	Communication line character parity.
prompt	Change the *Kermit* program's prompt.
receive	Set various parameters for inbound packets.
send	Set various parameters for outbound packets.
speed	Communication line speed.

The 'remote' commands are:

cwd	Change remote working directory.
delete	Delete remote files.

directory	Display a listing of remote file names.
help	Request help from a remote server.
host	Issue a command to the remote host in its own command language.
space	Display current disk space usage on remote system.
type	Display a remote file on your screen.
who	Display who's logged in, or get information about a user.

FILES

$HOME/.kermrc	*Kermit* initialization commands
./.kermrc	more *Kermit* initialization commands

SEE ALSO

cu(1C), uucp(1C)
Frank da Cruz and Bill Catchings, *Kermit User's Guide*, Columbia University, 6th Edition

DIAGNOSTICS

The diagnostics produced by *Kermit* itself are intended to be self-explanatory.

BUGS

See recent issues of the Info-Kermit digest (on ARPANET or Usenet), or the file ckuker.bwr, for a list of bugs.

NAME

lam – laminate files

SYNOPSIS

lam [–[**fp**] min.max] [–s sepstring] [–t c] file ...

DESCRIPTION

Lam copies the named files side by side onto the standard output. The *n*-th input lines from the input *file*s are considered fragments of the single long *n*-th output line into which they are assembled. The name '–' means the standard input, and may be repeated.

Normally, each option affects only the *file* after it. If the option letter is capitalized it affects all subsequent files until it appears again uncapitalized. The options are described below.

–f min.max
: Print line fragments according to *min.max*, where *min* is the minimum field width and *max* the maximum field width. If *min* begins with a zero, zeros will be added to make up the field width, and if it begins with a '–', the fragment will be left-adjusted within the field.

–p min.max
: Like **–f**, but pad this file's field when end-of-file is reached and other files are still active.

–s sepstring
: Print *sepstring* before printing line fragments from the next file. This option may appear after the last file.

–t c
: The input line terminator is *c* instead of a newline. The newline normally appended to each output line is omitted.

To print files simultaneously for easy viewing use *pr*(1).

EXAMPLES

The command

```
lam file1 file2 file3 file4
```

joins 4 files together along each line. To merge the lines from four different files use

```
lam file1 -S "\
" file2 file3 file4
```

Every 2 lines of a file may be joined on one line with

```
lam - - < file
```

and a form letter with substitutions keyed by '@' can be done with

```
lam -T @ letter changes
```

SEE ALSO

pr(1), join(1), printf(3)

AUTHOR

John Kunze

BUGS

NAME

mkmf – makefile editor

SYNOPSIS

mkmf [**-acdil**] [-f makefile] [-F template] [macroname=value ...]

DESCRIPTION

Mkmf creates a makefile that tells the *make* command how to construct and maintain programs and libraries. After gathering up all the source code file names in the current working directory and inserting them into the makefile, *mkmf* scans source code files for included files and generates dependency information which is appended to the makefile. Source code files are identified by their file name suffixes. *Mkmf* knows about the following suffixes:

.c	C
.e	Efl
.F	Fortran
.f	Fortran
.h	Include files
.i	Pascal include files
.l	Lex or Lisp
.o	Object files
.p	Pascal
.r	Ratfor
.s	Assembler
.y	Yacc

Mkmf checks for an existing makefile before creating one. If no -f option is present, the makefiles 'makefile' and 'Makefile' are tried in order.

After the makefile has been created, arbitrary changes can be made using a regular text editor. *Mkmf* can also be used to re-edit the macro definitions in the makefile, regardless of changes that may have been made since it was created.

By default, *mkmf* creates a program makefile. To create a makefile that deals with libraries, the -l option must be used.

Make Requests

Given a makefile created by *mkmf, make* recognizes the following requests:

all	Compile and load a program or library.
clean	Remove all unnecessary files.
depend	Edit the makefile and regenerate the dependency information.
extract	Extract all the object files from the library and place them in the same directory as the source code files. The library is not altered.
index	Print an index of functions on standard output.
install	Compile and load the program or library and move it to its destination directory.
library	Compile and load a library.
print	Print source code files on standard output.
tags	Create a tags file for the *ex* editor, for C, Pascal, and Fortran source code files.
program	Compile and link a program.
update	Recompile only if there are source code files that are newer than the program or library, link and install the program or library. In the case of an out-of-

date library, all the object files are extracted from the library before any recompilation takes place.

Several requests may be given simultaneously. For example, to compile and link a program, move the program to its destination directory, and remove any unnecessary object files:

```
make program install clean
```

Macro Definitions

Mkmf understands the following macro definitions:

CFLAGS	C compiler flags. After searching for included files in the directory currently being processed, *mkmf* searchs in directories named in -I compiler options, and then in the '/usr/include' directory.
DEST	Directory where the program or library is to be installed.
EXTHDRS	List of included files external to the current directory. *Mkmf* automatically updates this macro definition in the makefile if dependency information is being generated.
FFLAGS	Fortran compiler flags. After searching for included files in the directory currently being processed, *mkmf* searchs in directories named in -I compiler options, and then in the '/usr/include' directory.
HDRS	List of included files in the current directory. *Mkmf* automatically updates this macro definition in the makefile.
LIBRARY	Library name. This macro also implies the -l option.
LIBS	List of libraries needed by the link editor to resolve external references.
MAKEFILE	Makefile name.
OBJS	List of object files. *Mkmf* automatically updates this macro definition in the makefile.
PROGRAM	Program name.
SRCS	List of source code files. *Mkmf* automatically updates this macro definition in the makefile.
SUFFIX	List of additional file name suffixes for *mkmf* to know about.

Both these and any other macro definitions already within the makefile may be replaced by definitions on the command line in the form *macroname=value* . For example, to change the C compiler flags, the program name, and the destination directory in the makefile, the user might type the following line:

```
mkmf "CFLAGS=-I../include -O" PROGRAM=mkmf DEST=/usr/new
```

Note that macro definitions like CFLAGS with blanks in them must be enclosed in double quote '"' marks.

File Name Suffixes

Mkmf can be instructed to recognize additional file name suffixes, or ignore ones that it already knows about, by specifying suffix descriptions in the SUFFIX macro definition. Each suffix description takes the form '*.suffix:tI*' where *t* is a character indicating the contents of the file (s = source file, o = object file, h = header file, x = executable file) and *I* is an optional character indicating the include syntax for included files (C = C syntax, F = Fortran, Efl, and Ratfor syntax, P = Pascal syntax). The following table describes the default configuration for *mkmf*:

.c:sC	C

.e:sF	Efl
.F:sF	Fortran
.f:sF	Fortran
.h:h	Include files
.i:h	Pascal include files
.l:sC	Lex or Lisp
.o:o	Object files
.p:sP	Pascal
.r:sF	Ratfor
.s:s	Assembler
.y:sC	Yacc

For example, to change the object file suffix to .obj, undefine the Pascal include file suffix, and prevent Fortran files from being scanned for included files, the SUFFIX macro definition might look like:

"SUFFIX = .obj:o .i: .f:s"

Include Statement Syntax

The syntax of include statements for C, Fortran, and Pascal source code are of the form

C: #include "*filename*"
where # must be the first character in the line.

Fortran:
include '*filename*'
INCLUDE '*filename*'
where the include statement starts in column 7.

ascal: #include "*filename*"
#INCLUDE "*filename*"
where # must be the first character in the line.

User-Defined Templates

If *mkmf* can not find a makefile within the current directory, it normally uses one of the standard makefile templates, 'p.Makefile' or 'l.Makefile', in /usr/new/lib unless the user has alternative 'p.Makefile' or 'l.Makefile' template files in a directory $PROJECT/lib where $PROJECT is the absolute pathname of the directory assigned to the PROJECT environment variable.

OPTIONS

-a When searching a directory for source and include files, also consider files which have names beginning with periods. By default, *mkmf* ignores file names which have leading "dots," such as those of backup files created by some editors.

-c Suppress 'creating *makefile* from ...' message.

-d Turn off scanning of source code for 'include' files. Old dependency information is left untouched in the makefile.

-f *makefile*
Specify an alternative *makefile* file name. The default file name is 'Makefile'.

-i Cause *mkmf* to prompt the user for the name of the program or library, and the directory where it is to be installed. If a carriage return is typed in response to each of these queries, *mkmf* will assume that the default program name is *a.out* or the default library name is *lib.a,* and the destination directory is the current directory.

-l Force the makefile to be a library makefile.

−F *template*
Specify an alternative makefile *template* file name. The default program makefile template is ‘p.Makefile’ and the default library makefile template is ‘l.Makefile’. *Mkmf* normally looks for *template* in /usr/new/lib or $PROJECT/lib. However, *template* can be specified as an absolute pathname.

FILES

/usr/new/lib/p.Makefile	Standard program makefile template.
/usr/new/lib/l.Makefile	Standard library makefile template.
$PROJECT/lib/p.Makefile	User-defined program makefile template.
$PROJECT/lib/l.Makefile	User-defined library makefile template.

SEE ALSO
ar(1), ctags(1), ex(1), ld(1), ls(1), make(1)

Feldman, S.I., "Make – A Program for Maintaining Computer Programs"

Walden, K., "Automatic Generation of Make Dependencies", *Software–Practice and Experience*, vol. 14, no. 6, pp. 575-585, June 1984.

DIAGNOSTICS
Exit status 0 is normal. Exit status 1 indicates an error.

AUTHOR
Peter J. Nicklin

BUGS
The name of the makefile is included as a macro definition within the makefile and must be changed if the makefile is renamed.

Since executable files are dependent on libraries, standard library abbreviations must be expanded to full pathnames within the LIBS macro definition in the makefile.

Generated dependency information appears after a line in the makefile beginning with ‘###’. This line must **not** be removed, nor must any other information be inserted in the makefile below this line.

NAME

pathalias, makedb – electronic address router

SYNOPSIS

pathalias [**-ivc**] [**-t** *link*] [**-l** *host*] [**-d** *link*] [*files*]

makedb [**-a**] [**-o** *dbmfile*] [*files ...*]

DESCRIPTION

pathalias computes the shortest paths and corresponding routes from one host (computer system) to all other known, reachable hosts. *pathalias* reads host-to-host connectivity information on standard input or in the named *files*, and writes a list of host-route pairs on the standard output.

makedb takes *pathalias* output and creates or appends to a *dbm*(3) database.

Here are the *pathalias* options:

-i Ignore case: map all host names to lower case. By default, case is significant.

-c Print costs: print the path cost (see below) before each host-route pair.

-v Verbose: report some statistics on the standard error output.

-l *host*
Set local host name to *host*. By default, *pathalias* discovers the local host name in a system-dependent way.

-d *arg*
Declare a dead link, host, or network (see below). If *arg* is of the form "host1!host2," the link from host1 to host2 is treated as an extremely high cost (*i.e.*, DEAD) link. If *arg* is a single host name, that host is treated as dead and is be used as an intermediate host of last resort on any path. If *arg* is a network name, the network requires a gateway.

-t arg
Trace input for link, host or network on the standard error output. The form of *arg* is as above.

Here are the *makedb* options:

-a Append to an existing database; by default, *makedb* truncates the database.

-o *dbmfile*
Identify the output file base name.

pathalias Input Format

A line beginning with white space continues the preceding line. Anything following '#' on an input line is ignored.

A list of host-to-host connections consists of a "from" host in column 1, followed by white space, followed by a comma-separated list of "to' hosts, called *links*. A link may be preceded or followed by a network character to use in the route. Valid network characters are '!' (default), '@', ':', and '%'. A link (and network character, if present) may be followed by a "cost" enclosed in parentheses. Costs may be arbitrary arithmetic expressions involving numbers, parentheses, '+', '–', '*', and '/'. The following symbolic costs are recognized:

LOCAL	25	(local-area network connection)
DEDICATED	95	(high speed dedicated link)
DIRECT	200	(toll-free call)
DEMAND	300	(long-distance call)
HOURLY	500	(hourly poll)
EVENING	1800	(time restricted call)
DAILY	5000	(daily poll, also called POLLED)

WEEKLY	30000	(irregular poll)

In addition, DEAD is a very large number (effectively infinite), and HIGH and LOW are -5 and +5 respectively, for baud-rate or quality bonuses/penalties. These symbolic costs represent an imperfect measure of bandwidth, monetary cost, and frequency of connections. For most mail traffic, it is important to minimize the number of intermediaries in a route, thus, *e.g.*, HOURLY is far greater than DAILY / 24. If no cost is given, a default of 4000 is used.

For the most part, arithmetic expressions that mix symbolic constants other than HIGH and LOW make no sense. *E.g.*, if a host calls a local neighbor whenever there is work, and additionally polls every evening, the cost is DIRECT, **not** DIRECT+EVENING.

Some examples:

```
down          princeton!(DEDICATED), tilt,
              %thrash(LOCAL)
princeton     topaz!(DEMAND+LOW)
topaz         @rutgers(LOCAL)
```

If a link is encountered more than once, the least-cost occurrence dictates the cost and network character. Links are treated as bidirectional, to the extent that a DEAD reverse link is assumed unless better information is available.

The set of names by which a host is known by its neighbors is called its *aliases*. Aliases are declared as follows:

```
name = alias, alias ...
```

The name used in the route to or through aliased hosts is the name by which the host is known to its predecessor in the route.

Fully connected networks, such as the ARPANET or a local-area network, are declared as follows:

```
net = {host, host, ...}
```

The host-list may be preceded or followed by a routing character, and may be followed by a cost:

```
princeton-ethernet = {down, up, princeton}!(LOCAL)
ARPA = @{sri-unix, mit-ai, su-score}(DEDICATED)
```

See also the sections on *gateways* and *domains* below.

Connection data may be given while hiding host names by declaring

```
private {host, host, ...}
```

pathalias will not generate routes for private hosts, but may produce routes through them. The scope of a private declaration extends from the declaration to the end of the input file in which it appears. It is best to put private declarations at the beginning of the appropriate input file.

Output Format
A list of host-route pairs is written to the standard output, where route is a string appropriate for use with *printf*(3), *e.g.*,

```
rutgers       princeton!topaz!%s@rutgers
```

The "%s" in the route string should be replaced by the user name at the destination host. (This task is normally performed by a mailer.)

Except for *domains* (see below), the name of a network is never used in expansions. Thus, in the earlier example, the path from down to up would be "up!%s," not "princeton-ethernet!up!%s."

Gateways
A network is represented by a pseudo-host and a set of network members. Links from the members to the network have the weight given in the input, while the cost from the network to the members is zero. If a network is declared dead on the command line (with the **-d** option), the member-to-network links are marked dead, which discourages paths to members by way of the network.

If the input also shows a link from a host to the network, then that host will be preferred as a gateway. Gateways need not be network members.

E.g., suppose CSNET is declared dead on the command line and the input contains

```
CSNET = {...}
csnet-relay          CSNET
```

Then routes to CSNET hosts will use csnet-relay as a gateway.

pathalias discourages forwarding beyond dead networks.

Domains
A host or network whose name begins with '.' is called a domain. Domains are presumed to require gateways, *i.e.*, they are DEAD. The route given by a path through a domain is similar to that for a network, but here the domain name is tacked onto the end of the next host. Subdomains are permitted. *E.g.*,

```
harvard              .EDU
.EDU = {.BERKELEY}
.BERKELEY            ernie
```

yields

```
ernie                ...!harvard!ernie.BERKELEY.EDU!%s
```

Output is given for the nearest gateway to a domain, *e.g.*, the example above gives

```
.EDU                 ...!harvard!%s
```

Output is given for a subdomain if it has a different route than its parent domain, or if all of its ancestor domains are private.

Databases
Makedb builds a *dbm*(3) database from the standard input or from the named *files*. (*Makedb* replaces the obsolete **-b** option of *pathalias*, which is no longer recognized.) Input is expected to be sequence of ASCII records, each consisting of a key field and a data field separated by a single tab. If the tab is missing, the data field is assumed to be empty.

FILES ET AL.

/usr/local/lib/palias.{dir,pag}	default dbm output
newsgroup mod.map	likely location of some input files

getopt(3), available from newsgroup mod.sources (if not in the C library).

BUGS

The order of arguments is significant. In particular, **-i** and **-t** should appear early.

pathalias can generate hybrid (*i.e.* ambiguous) routes, which are abhorrent and most certainly should not be given as examples in the manual entry.

Multiple '@'s in routes are prohibited by many mailers, so *pathalias* resorts to the "magic %" rule when appropriate. This convention is not documented anywhere, including here.

Domains constitute a futile attempt to defeat anarchy and otherwise retard progress.

AUTHORS
Steve Bellovin (ulysses!smb)
Peter Honeyman (princeton!honey)

NAME

rs – reshape a data array

SYNOPSIS

rs [–[csCS][x][kKgGw][N]tTeEnyjhHm] [rows [cols]]

DESCRIPTION

Rs reads the standard input, interpreting each line as a row of blank-separated entries in an array, transforms the array according to the options, and writes it on the standard output. With no arguments it transforms stream input into a columnar format convenient for terminal viewing.

The shape of the input array is deduced from the number of lines and the number of columns on the first line. If that shape were inconvenient, a more useful one might be obtained by skipping some of the input with the **–k** option. Other options control interpretation of the input columns.

The shape of the output array is influenced by the *rows* and *cols* specifications, which should be positive integers. If only one of them is a positive integer, *rs* computes a value for the other which will accommodate all of the data. When necessary, missing data are supplied in a manner specified by the options and surplus data are deleted. There are options to control presentation of the output columns, including transposition of the rows and columns.

The options are described below.

–cx	Input columns are delimited by the single character *x*. A missing *x* is taken to be '^I'.
–sx	Like –**c**, but maximal strings of *x* are delimiters.
–Cx	Output columns are delimited by the single character *x*. A missing *x* is taken to be '^I'.
–Sx	Like –**C**, but padded strings of *x* are delimiters.
–t	Fill in the rows of the output array using the columns of the input array, that is, transpose the input while honoring any *rows* and *cols* specifications.
–T	Print the pure transpose of the input, ignoring any *rows* or *cols* specification.
–kN	Ignore the first *N* lines of input.
–KN	Like **–k**, but print the ignored lines.
–gN	The gutter width (inter-column space), normally 2, is taken to be *N*.
–GN	The gutter width has *N* percent of the maximum column width added to it.
–e	Consider each line of input as an array entry.
–n	On lines having fewer entries than the first line, use null entries to pad out the line. Normally, missing entries are taken from the next line of input.
–y	If there are too few entries to make up the output dimensions, pad the output by recycling the input from the beginning. Normally, the output is padded with blanks.
–h	Print the shape of the input array and do nothing else. The shape is just the number of lines and the number of entries on the first line.
–H	Like **–h**, but also print the length of each line.
–j	Right adjust entries within columns.
–wN	The width of the display, normally 80, is taken to be the positive integer *N*.
–m	Do not trim excess delimiters from the ends of the output array.

With no arguments, *rs* transposes its input, and assumes one array entry per input line unless the first non-ignored line is longer than the display width. Option letters which take numerical arguments interpret a missing number as zero unless otherwise indicated.

EXAMPLES

Rs can be used as a filter to convert the stream output of certain programs (e.g., *spell*, *du*, *file*, *look*, *nm*, *who*, and *wc*(1)) into a convenient "window" format, as in

```
who | rs
```

This function has been incorporated into the *ls*(1) program, though for most programs with similar output *rs* suffices.

To convert stream input into vector output and back again, use

```
rs 1 0 | rs 0 1
```

A 10 by 10 array of random numbers from 1 to 100 and its transpose can be generated with

```
jot -r 100 | rs 10 10 | tee array | rs -T > tarray
```

In the editor *vi*(1), a file consisting of a multi-line vector with 9 elements per line can undergo insertions and deletions, and then be neatly reshaped into 9 columns with

```
:1,$!rs 0 9
```

Finally, to sort a database by the first line of each 4-line field, try

```
rs -eC 0 4 | sort | rs -c 0 1
```

SEE ALSO

jot(1), vi(1), sort(1), pr(1)

AUTHOR

John Kunze

BUGS

Handles only two dimensional arrays.

The algorithm currently reads the whole file into memory, so files that do not fit in memory will not be reshaped.

Fields cannot be defined yet on character positions.

Re-ordering of columns is not yet possible.

There are too many options.

NAME

tac – concatenate and print files in reverse

SYNOPSIS

tac [**–string**] [**+string**] file ...

DESCRIPTION

Tac reads each *file* in sequence and writes it on the standard output, reversed by the file segments delimited by *string*. *–string* specifies segments bounded on the left by *string,* while *+string* specifies right-bounded segments. The default is +\n (print lines in reverse order).

EXAMPLES

```
tac '-\
From ' /usr/spool/mail/$USER
```

prints out one's mail messages, most recent first.

```
tac file
```

prints the file in reverse, line by line, and:

```
tac file1 file2 >file3
```

reverses each of the first two files by line and places the concatenated result on the third.

SEE ALSO

cat(1), rev(1), tail(1), tmail(1)

BUGS

Tac doesn't yet handle multiple argument files exactly right. It's also unclear which direction it should process them in.
Tac does not (and cannot efficiently) work on piped input.

NAME

tmail – print out mail messages, most recent first

SYNOPSIS

tmail [username] [mboxfile]

DESCRIPTION

Tmail prints Unix-style mail messages in reverse order (most recent first). If no argument is given it looks in your system maildrop (*/usr/spool/mail/$USER*). An argument which is a valid *username* causes *tmail* to look in that person's maildrop; otherwise the argument should be the name of a Unix-style "mailbox" file.

SEE ALSO

tac(1), cat(1).

BUGS

Should handle multiple arguments.

NAME

umodem - Version 3.1 -- UNIX-Based Remote File Transfer Facility

SYNOPSIS

Usage:
  umodem -[c!rb!rt!sb!st][options] filename

Major Commands --
  c <-- Enter Command Mode
  rb <-- Receive Binary
  rt <-- Receive Text
  sb <-- Send Binary
  st <-- Send Text
Options --
  1 <-- (one) Employ TERM II FTP 1
  4 <-- Enable TERM FTP 4
  7 <-- Enable 7-bit transfer mask
  a <-- Turn ON ARPA Net Flag
  d <-- Do not delete umodem.log file before starting
  l <-- (ell) Turn OFF LOG File Entries
  m <-- Allow file overwiting on receive
  p <-- Turn ON Parameter Display
  y <-- Display file status (size) information only

DESCRIPTION

Umodem uses the Christensen protocol to transfer files to and from CP/M systems.

Umodem -- Implements the "CP/M User's Group XMODEM" protocol, the TERM II File Transfer Protocol (FTP) Number 1, and the TERM II File Transfer Protocol Number 4 for packetized file up/downloading.

There is currently no batch transfer capability. The program writes logging data to a file in the user's home directory called umodem.log.

The program will do a protocol file transfer with error checking to or from a CP/M system running Ward Christensen's program MODEM or one of its derivatives (MODEM7 or APMOD777 etc.) or any program that uses the same protocols (e.g. ZPRO, TERM II). Note that executable and squeezed files must use the -sb or -rb options.

Umodem supports an interactive mode in which the user may perform a number of Umodem-oriented functions without leaving Umodem. These functions (and their commands) are:

UMODEM COMMAND MODE OPTIONS

Usage: r or s or option
Major Commands --
  rb <-- Receive Binary
  rt <-- Receive Text
  sb <-- Send Binary
  st <-- Send Text
Options --
  1 <-- (one) Employ TERM II FTP 1

```
        3  <-- Enable TERM FTP 3 (CP/M UG)
        7  <-- Toggle 7-bit transfer mask
        a  <-- Turn ON ARPA Net Flag
        l  <-- Toggle LOG File Entries
        m  <-- Allow file overwiting on receive
        x  <-- Exit
        y  <-- Display file status (size) information only
```

UMODEM COMMAND MODE

The following is a sample session illustrating what can be done in the command mode of Umodem.

```
$ umodem -c

UMODEM Version 3.5 -- UNIX-Based Remote File Transfer Facility

UMODEM:  LOG File '/user/rxc/umodem.log' is Open

UMODEM Command Mode -- Type ? for Help
3  L  UMODEM> ?

Usage: r or s or option
Major Commands --
        rb <-- Receive Binary
        rt <-- Receive Text
        sb <-- Send Binary
        st <-- Send Text
Options --
        1  <-- (one) Employ TERM II FTP 1
        3  <-- Enable TERM FTP 3 (CP/M UG)
        7  <-- Enable 7-bit transfer mask
        a  <-- Turn ON ARPA Net Flag
        l  <-- Toggle LOG File Entries
        m  <-- Allow file overwiting on receive
        x  <-- Exit
        y  <-- Display file status (size) information only

3  L  UMODEM> 1

TERM FTP 1 Selected
1  L  UMODEM> m

File Overwriting  Enabled
1  LM UMODEM> m

File Overwriting NOT Enabled
1  L  UMODEM> 7

7-Bit Transfer  Selected
17 L  UMODEM> 7

7-Bit Transfer NOT Selected
```

```
1 L UMODEM> y umodem.c

UMODEM File Status Display for umodem.c
  Estimated File Size 42K, 331 Records, 42252 Bytes

1 L UMODEM> x
```

FILES

umodem.log keeps a log of transfers to and from and any problems during transfer.

SEE ALSO

AUTHOR

- Lauren Weinstein, 6/81
- (Version 2.0) Modified for JHU/UNIX by Richard Conn, 8/1/81
- Version 2.1 Mods by Richard Conn, 8/2/81
- Version 2.2 Mods by Richard Conn, 8/2/81
- Version 2.3 Mods by Richard Conn, 8/3/81
- Version 2.4 Mods by Richard Conn, 8/4/81
- Version 2.5 Mods by Richard Conn, 8/5/81
- Version 2.6 Mods by Bennett Marks, 8/21/81 (Bucky @ CCA-UNIX)
- Version 2.7 Mods by Richard Conn, 8/25/81 (rconn @ BRL)
- Version 2.8 Mods by Richard Conn, 8/28/81
- Version 2.9 Mods by Richard Conn, 9/1/81
- Version 3.0 Mods by Lauren Weinstein, 9/14/81
- Version 3.1 Mods by Lauren Weinstein, 4/17/82
- Version 3.2 Mods by Michael M Rubenstein, 5/26/83
- Version 3.3 Mod by Ben Goldfarb, 07/02/83
- Version 3.4 Mods by David F. Hinnant, NCECS, 7/15/83
- Version 3.5 Mods by Richard Conn, 08/27/83

UMODEM.MAN – Received From:

```
Received: From Ucb-Vax.ARPA by BRL via smtp;  6 Sep 83 9:11 EDT
Received: by ucbvax.ARPA (4.9/4.7)
        id AA21658; Tue, 6 Sep 83 06:11:52 PDT
Message-Id: <8309061311.AA21658@ucbvax.ARPA>
Date: 5-Sep-83 20:22:19-PDT (Mon)
Original-From: ucsfpgs!brian (#Brian Katzung)
From: ucsfcgl!ucsfpgs!brian@Berkeley (#Brian Katzung)
Subject: umodem.1
To: rconn@brl.ARPA
```

UMODEM.1 – Modified by:

Richard Conn

ANSI
Tom Quarles, Berkeley

NAME

ansitape - ANSI standard tape handler

SYNOPSIS

ansitape [key] [keyargs] [files]

DESCRIPTION

Ansitape reads and writes magnetic tapes written in ANSI standard format (called "Files-11" by DEC). Tapes written by *ansitape* are labeled with the first 6 characters of the machine name by default. Actions are controlled by the *key* argument. The *key* is a string of characters containing at most one function letter. Other arguments to the command are a tape label and file names specifying which files are to be written onto or extracted from the tape.

The function portion of the key is specified by one of the following letters:

r The named files are written at the end of the tape. The **c** function implies this.

x The named files are extracted from the tape. If no file argument is given, the entire contents of the tape is extracted. Note that if the tape has duplicated file names, only the last file of a given name can be extracted.

t The names of the specified files are listed each time they occur on the tape. If no file argument is given, all files on the tape are listed.

c Create a new tape; writing begins at the beginning of the tape instead of after the last file. This command implies **r.**

The following characters may be used in addition to the letter which selects the function desired.

f This argument allows the selection of a different tape device. The next word in the keyargs list is taken to be the full name of a device to write the tape on. The default is /dev/rmt12.

n The **n** option allows the user to specify as the next argument in the keyargs list, a control file containing the names of files to put on the tape. If the file name is '-', the control file will, instead, be read from standard input. The control file contains one line for each file to be placed on the tape. Each line has two names, the name of the file on the local machine, and the name it is to have when placed on the tape. This allows for more convenient flattening of hierarchies when placing them on tape. If the second name is omitted, the UNIX file name will be used on the tape also. This argument can only be used with the **r** and **c** functions.

l The **l** option allows the user to specify the label to be placed on the tape. The next argument in the keyargs list is taken as the tape label, which will be space padded or truncated to six characters. This option is meaningless unless **c** is also specified.

v Normally *ansitape* works relatively silently. The **v** (verbose) option causes it to type information about each file as it processes it.

b The **b** option allows the user to select the blocksize to be used for the tape. By default, *ansitape* uses the maximum block size permitted by the ANSI standard, 2048. Some systems will permit a much large block size, and if large files are being put on the tape it may be advantageous to do so. *Ansitape* will take the next argument of the keyargs list as the blocksize for the tape. Values below 18 or above 32k will be limited to that range. The standard scale factors b=512 and k=1024 are accepted.

Ansitape will not copy directories, character or block special files, symbolic links, sockets, or binary executables. Attempts to put these on tape will result in warnings, and they will be skipped completely.

FILES

/dev/rmt12

DIAGNOSTICS

A warning message will be generated when a record exceeds the maximum record length and the affected file will be truncated.

BUGS

Ansitape quietly truncates names longer than 17 characters.
ANSI 'f' format files can be read but not written.
Multivolume tapes can not be handled.

NAME

vmsprep - VMS tape preperation aid

SYNOPSIS

vmsprep [-] [name ...]

DESCRIPTION

Vmsprep traverses hierarchies of files and prepares them for transportation to VMS. Since ANSI stardard tapes (the VMS standard) do not allow hierarchy, this program provides a method of flattening the hierarchy onto a tape in such a way that it can be unpacked on VMS to recreate the same tree structure.

For reasons best not described here, *vmsprep* will attempt to exclude all RCS and SCCS archives by ignoring all files or directories named 'RCS' or 'SCCS', or files starting with 's.' or ending in ',v'.

The output of *vmsprep* is a pair of files vmsprep.namelist and UNPACK.COM. vmsprep.namelist is a list of files to be placed on the tape in the format required by *ansitape.* If the first argument is '-' instead of a file or directory name, vmsprep will instead send the namelist to standard output, and place UNPACK.COM in /tmp to avoid attempting to write in the current directory. All of the files except UNPACK.COM will be placed on the tape under cryptic names. UNPACK.COM is a VMS command script which will recreate all of the necessary directories and then move the cryptically named files to their proper place.

A typical sequence would be:

```
        vmsprep - tree1 tree2 file | ansitape cln trees -
```

Then on a VMS machine

```
        mount MFA0: trees
        copy MFA0:*.*.* *
        @UNPACK
```

FILES

vmsprep.namelist
UNPACK.COM

DIAGNOSTICS

A warning is reported if a file or directory name contains a character not permitted in VMS names. The offending character is replaced by 'Z' and *vmsprep* continues.

SEE ALSO

ansitape(l)

BUGS

Extra periods in file names may not be dealt with optimally.
All files and directories to be moved must be descendants of the current working directory.
Absolute path names and paths containing ".." will produce unpredictable results.
Since vmsprep uses find(1) internally, it does not follow symbolic links.
The exclusion of RCS and SCCS files should be controlled by a command line flag.
Assumes VMS v4.0 or greater for long file names.

APL SYSTEM
Purdue

NAME

apl – apl interpreter

SYNPOSIS

apl [–m] [–e] [–q] [–r] [–t] [–c] [–C] [–d] [–D] [ws]
apl2 [–m] [–e] [–q] [–r] [–t] [–c] [–C] [–d] [–D] [ws]

DESCRIPTION

This is the Unix APL interpreter. It has lived through several different versions of Unix and grown steadily more complex. Currently, a version of APL for Unix on the PDP-11 and the VAX is supported. This version supports monadic and dyadic domino, a state indicator of sorts, and Unix I/O quad functions.

The best documentation concerning the use of APL once it has been started from the shell is the *Unix APL\11 User's Manual.* This manual includes a list of the APL character set, system commands, quad functions, and i-beam functions, as well as an overall description of the use of APL. The specifics are contained in the four appendices for easy reference by the more experienced user.

The command invoking APL may optionally contain the name of a workspace file to be loaded (default is "continue", or, if "continue" does not exist in the current directory, APL starts executing with a "clear ws").

There are all sorts of flags which may be specified when APL is invoked. Only a subset of these are of general usefulness; the remainder exist for convenience in debugging and software maintenance purposes. In the following description, the flags are presented from those which are of the most general interest to those which are of interest only to persons maintaining APL.

Normally, APL runs in "ASCII mode". (This is discussed more fully following the description of the various flags.) If "–m" is specified, APL "maps" the standard input and standard output as appropriate for use with an APL terminal.

By default, APL attempts to determine whether or not the standard input is a terminal. If not, all input will be echoed to the standard output. In this fashion, when APL is run with a pipe or disc file as input, the output clearly shows the commands issued along with their results. The "–e" flag forces APL to echo its input to its output regardless of the input device. Similarly, "–q" ("quiet") forces APL not to echo its input to the standard output.

The flag "–r" has meaning only when the Purdue EE editor XED is used. This flag is passed by APL to XED to invoke funny XED stuff. This is generally a non-portable feature.

By default, APL places its scratch files into /tmp. If the "–t" flag is specified, temporary files will be placed into the current directory.

By default, APL catches fatal signals (e.g. memory fault, floating-point exception, etc.) and prints a termination message of the form:

fatal signal: message

It then exits normally. If the flag "–c" or "–C" is specified, it will print this error message and then exit via an "abort", producing a core dump. If the flag "–d" or "–D" is specified, it will not catch fatal errors, and thus will be automatically terminated by the Unix kernel if a fatal signal is received. (This will also invoke a core dump.) These flags are useful for debugging APL, but aren't of much use to the ordinary user.

The program "apl2" is identical to "apl" except that "apl" is double-precision and "apl2" is single-precision. Workspaces are stored in whatever precision is in use, and are converted if necessary automatically when they are ")load"ed. Effectively, "apl2" has twice as much space in its internal workspace.

APL is designed to operate principally from ASCII terminals. Upper-case letters are used for the various APL symbols, as described in a separate document. Overstrike characters, which generally will not appear as overstruck characters on a CRT screen, are generated by typing the first character, a control-H, and the second character. The order of the two characters is not significant. The workspace used by APL is stored in this special ASCII format.

APL does support APL terminals. To use APL from an APL terminal, it is necessary to specify the "–m" flag when calling APL from the shell; this causes the APL character set to be mapped to/from ASCII for input/output. The workspace file is still stored in ASCII format; thus work may be done interchangeably on both types of terminals.

HISTORY

APL was originally written at Bell Labs by Ken Thompson, sometime before version six Unix. It was modified for a while at Yale University, and then came to Purdue University, where it has undergone extensive modification. It is currently being supported by the Electrical Engineering Unix network. Complaints, suggestions, or whatever should be forwarded to user "bruner" on the EE Network system, or sent to either John Bruner or Dr. Anthony P. Reeves in the school of Electrical Engineering at Purdue University.

FILES

/tmp/apled.###### - editor temporary file
/tmp/aplws.###### - workspace temporary file
continue - default workspace file

SEE ALSO

aplcvt(1) – convert between PDP-11 and VAX workspace formats
aplopr(1) – output APL files to the Printronix printer
cata(1) – display functions with APL line numbers
prws(1) – print workspace

BUGS

Character comparisons do not work.
Only a restricted form of dyadic format is available. Laminate is not supported.
The workspace size on the PDP-11 is limited to about 5000 items in APL and 10000 in APL2.
The workspace size on the VAX is limited only by the virtual memory system.

NAME

aplcvt – convert APL workspaces between PDP-11 and VAX formats

SYNPOSIS

aplcvt [**–v**|**–p**][file ...]

DESCRIPTION

aplcvt performs the necessary transformations to produce a VAX format workspace from a PDP-11 format workspace, and *vice versa.* The workspace formats differ because the word sizes of the two machines are different; hence, pointers have different lengths. In general, any PDP-11 workspace can be converted to VAX format. VAX format workspaces can be converted to PDP-11 format provided they are "small enough" (if a VAX workspace is too large to be converted to PDP-11 format, it is also too large to run on the PDP-11 APL interpreter).

In the usual case, the workspaces to be converted are specified on the command line. The output files have the same names as the input files with a ".pdp" or ".vax" extension. (If the input file name ends with a ".pdp" or ".vax" extension, that extension will be stripped off first.) Alternately, *aplcvt* can be used as a filter. (If either the standard input or the standard output are directed to a tty, a syntax message is output. It is highly unlikely that a user will want a converted binary workspace printed on his or her terminal.)

The default direction of conversion is to convert to the format used by the host machine; i.e. on a PDP-11 the default is to convert VAX format to PDP-11 format. If desired, the direction may be specified explicitly by a "–p" or "–v" flag.

SEE ALSO

apl(1) – the APL interpreter

BUGS

Occasionally, *aplcvt* will bomb out with an error when none really occurred. This seems to be due to a bug in the standard I/O library.

NAME

xed – eXtended text EDitor – V7.15

SYNOPSIS

xed [**–!@abBcdefhiklmnoOpqrstvwy**] [name]

DESCRIPTION

Xed is the eXtended text EDitor.

If a *name* argument is given, *xed* simulates an **e** command (see below) on the named file; that is to say, the file is read into *xed*'s buffer so that it can be edited. After every 35 (default) commands have been executed, the edit buffer will be written on a scratch file. When *xed* terminates successfully, the *save* file will be removed unless the **–d** flag was selected. If a writeable file named "*edsav*" exists in the current directory, all commands typed will be written to it.

The optional flags after the – have the following functions:

–! Disallow use of the ! command. Mostly useful for writing programs which cannot allow unrestricted access to shell commands.

–@*fn* Preset the *indirect* file name to *fn*. Subsequent use of the @ command will read commands from *fn*, until the name is changed by giving an argument to the @ command.

–a The line numbers will be printed in *apl* mode. The form is "[*n*]*t*" followed by the text. In addition, overstruck characters will be printed on two lines, one above the other. *Apl* line numbers begin at *zero* instead of one.

–b Make a *backup* copy of the edit file upon entry to the editor. The file's name will be that of the original file with a "*.bak*" extension.

–B*nnnn*
Set the *line* buffer size to *nnnn* (decimal) bytes. The default line buffer size is 512 bytes, which limits the maximum length line which may be processed. Since there are occasions where it is desired to process longer lines, the buffer size may be increased.

–c*nn* Set the editor's idea of the depth of the Crt screen for the : command to *nn* (decimal). Default is 21 lines. If *nn* is zero, the paging will be disabled. (See also the *d=nn* command.)

–d Disables the deletion of the file created via the *auto-save* feature. (The "*.edt*" file.)

–e Each input command will be echoed on standard output. This is useful for debugging editor command files, since the error message will be immediately preceded by the command that caused it.

–f *Xed* will automatically prompt for text lines upon being invoked. Upon exit, *xed* will automatically write the file. This is useful for creating files without having to type the **a** command upon entry. Note: If this flag is selected, the editor will over-write an existing file by the same name. See the **qi** command.

–h Enable processing of a "*huge*" file, I.E. one with up to 511 blocks, instead of the normal limit of 255 blocks. The use of **–h** disallows the **g** and **v** commands. (This flag is inoperative and unnecessary on the Vax.)

–i If an *interrupt* (ASCII **DEL**) character is typed, *xed* will write the current contents of the edit buffer on a file, and exit. The name of the dump file is that of the original file with a *.int* extension. The **–i** flag is very useful for shell files which call the editor, since the editor will not hang around after an interrupt, interfering with the user's commands.

−k Useful for slow terminals, this flag *kills* verbose error messages. Instead, *xed* prints a query ? followed by an *error number*. The actual error message may be obtained by typing the e*nn* command (see below). The long error messages may be turned on/off via the **e+** and **e−** commands (see below).

−l*c* The *eol* character is initialized to character *c*. It may be changed during the edit session by the **e=***c* command.

−m*nn* The modification count before an automatic save of the edit buffer is set to *nn* (decimal). Default is 35. (That is, after every 35 commands which cause a modification to one or more lines, the edit buffer will be written on the edit file name with *.edt* extension.) If the count is zero, the auto-save feature is disabled.

−n The *no-line-numbers* flag is toggled. This results in the omission of line number prompts as well as line numbers on the **p** and **l** commands.

−o The editor will not seek standard input to end-of-file upon detecting a command error. Normally, this results in a command file terminating immediately.

−O If a *write* is attempted to a file that is write-locked, but is owned by the user, an attempt will be made to *override* the permission.

−p Turn on prompts even if not talking to a terminal, mostly useful for editing through pipes (as when using *protocol*(1) or *script*(1)).

−q The editor will NOT ignore a *quit* (ASCII FS or ctrl-\) signal. Normally for editor debugging purposes, as a core dump can then be made.
Beware, the edit buffer can not be recovered!

−r *Removes* the special meaning of the special characters: **$ & \(\) [. * ^ **

−s *Silent* mode. No prompts are issued, printing of lines resulting from commands is suppressed unless they are *explicitly* terminated with a **p**. This mode is useful for running editor command files.

−t*c* Set the *tab* character to *c*. This is the character which will be expanded to the appropriate number of fill characters to get to the next column which has a tab stop set in it. The *tab* character may be set/changed using the **t=***c* command.

−v*c* Set the tab *fill* character to *c*. This character is used to pad out the space between expanded fields. The tab *fill* character may be set/changed by the **f=***c* command.

−w*nn* Set the editor's idea of the page width to *nn* (decimal). Default is 80 columns. (See also the **w=***nn* command.)

−y Set the interrupt processing to list out one page (see the : command) upon receipt of an interrupt.

−0123456789
A decimal number preceded by a − will set a *tab* stop in that column. Tab settings may be made during edit session by the **t,***nn* command.

−, A comma in the flag list is ignored to facilitate setting multiple tab stops. For example, tabs may be set by any of the forms "−9 −17 −25", "−9,17,25", "−9a17d25f".

Xed operates on a copy of any file it is editing; changes made in the copy have no effect on the file until a *w* (write) command is given. The copy of the text being edited resides in a temporary file called the buffer. There is only one buffer.

Commands to *xed* have a simple and regular structure: zero or more addresses followed by a one or more character command, possibly followed by parameters to the command. These addresses specify one or more lines in the buffer. Every command which requires addresses has default addresses, so that the addresses can often be omitted.

In general, only one command may appear on a line. (See the **e=***c* command and the **−l** flag.) Certain commands allow the input of text. This text is placed in the appropriate place in the buffer. While *xed* is accepting text, it is said to be in *input mode.* In this mode, no commands are recognized; all input is merely collected. Input mode is left by typing a period **.** alone at the beginning of a line, or by receipt of an end-of-file (Ctrl-D) from the keyboard.

Xed supports a limited form of *regular expression* notation. A regular expression specifies a set of strings of characters. A member of this set of strings is said to be *matched* by the regular expression. The regular expressions allowed by *xed* are constructed as follows: In the following specification for regular expressions the word *character* means any character but newline.

1. Any character except a *special* character matches itself. Special characters are the regular expression delimiter plus **\ [.** and sometimes **^ * $.**
2. A **.** matches any character.
3. A **** followed by any character except a *digit* or **( )** matches that character.
4. A nonempty string *s* bracketed **[***s***]** (or **[^***s***]**) matches any character in (or not in) *s*. In *s*, **** has no special meaning, and **]** may only appear as the first letter. A substring *a−b*, with *a* and *b* in ascending ASCII order, stands for the inclusive range of ASCII characters.
5. A regular expression of form 1-4 followed by ***** matches a sequence of *zero* or more matches of the regular expression.
6. A regular expression, *x*, of form 1-8, bracketed **\(***x***\)** matches what *x* matches, with side-effects described under the **s** command below.
7. A **** followed by a digit *n* matches a copy of the string that the bracketed regular expression beginning with the *n*th **\(** matched.
8. A regular expression of form 1-8, *x*, followed by a regular expression of form 1-7, *y* matches a match for *x* followed by a match for *y*, with the *x* match being as long as possible while still permitting a *y* match.
9. A regular expression of form 1-8 preceded by **^** (or followed by **$**), is constrained to matches that begin at the left (or end at the right) end of a line.
10. A regular expression of form 1-9 picks out the longest among the leftmost matches in a line.
11. An empty regular expression stands for a copy of the last regular expression encountered.

Regular expressions are used in addresses to specify lines and in one command (see **s** below) to specify a portion of a line which is to be replaced. If it is desired to use one of the regular expression metacharacters as an ordinary character, that character may be preceded by ****. This also applies to the character bounding the regular expression (often **/**) and to **** itself.

To understand addressing in *xed* it is necessary to know that at any time there is a *current line*. Generally speaking, the current line is the last line affected by a command; however, the exact effect on the current line is discussed under the description of the command. Addresses are constructed as follows.

1. The character **.** addresses the current line.
2. The character **$** addresses the last line of the buffer.
3. A decimal number *n* addresses the *n*-th line of the buffer.
4. **´***x* addresses the line (or lines) marked with the mark name character *x*, which must be a lower-case letter. An alternative to this syntax is the capital letter alone. Lines are

marked with the **k** command described below.

5. '*x*^ (or *X*^) addresses the first (lower) line of the range marked with the mark name character *x*. (See the *k* command description.)

6. '*x*$ (or *X*$) addresses the last (upper) line of the range marked with the mark name character *x*. (See the **k** command description.)

7. A regular expression enclosed in slashes / addresses the first line found by searching toward the end of the buffer and stopping at the first line containing a string matching the regular expression. If necessary the search wraps around to the beginning of the buffer.

8. A regular expression enclosed in queries ? addresses the first line found by searching toward the beginning of the buffer and stopping at the first line containing a string matching the regular expression. If necessary the search wraps around to the end of the buffer.

9. An address followed by a plus sign + or a minus sign − followed by a decimal number specifies that address plus (resp. minus) the indicated number of lines. The plus sign may be omitted.

10. If an address begins with + or − the addition or subtraction is taken with respect to the current line; e.g. **−5** is understood to mean **.−5**. (If the first address is omitted, but a second bound is specified, then the first address will be the current line plus one. e.g. "**,+10**" is equivalent to "**.+1,.+10**".)

11. If an address ends with + or −, then 1 is added (resp. subtracted). As a consequence of this rule and rule 10, the address − refers to the line before the current line. Moreover, trailing + and − characters have cumulative effect, so −− refers to the current line less 2. (There are complications of this rule, see the **b** command below.)

12. To maintain compatibility with earlier versions of the editor, the character ^ in addresses is entirely equivalent to −.

13. The character = specifies that the address bounds of the previous command are to be used for the current command.

14. The character pair =^ addresses the lower bound (first address) specified in the previous command.

15. The character pair =$ addresses the upper bound (second address) specified in the previous command.

16. The character pair .. addresses the last value of . different from the current value of ..

Commands may require zero, one, or two addresses. Commands which require no addresses regard the presence of an address as an error. Commands which accept one or two addresses assume default addresses when insufficient are given. If more addresses are given than such a command requires, the last one or two (depending on what is accepted) are used.

Addresses are separated from each other typically by a comma **,**. They may also be separated by a semicolon **;**. In this case the current line **.** is set to the first address before the next address is interpreted. The second address of any two-address sequence must correspond to a line following the line corresponding to the first address.

In the following list of *xed* commands, the default addresses are shown in parentheses. The parentheses are not part of the address, but are used to show that the given addresses are the default.

As mentioned, it is generally illegal for more than one command to appear on a line. However, most commands may be suffixed by **p**, **b**, **q** or **l**, in which case the current line is either printed (as in the **p** command), listed with balanced pairs of parentheses, square brackets, and brace brackets numbered (**b**), quoted (by " or ') string lengths (**q**), or listed as in the l command.

(.)**a**
text
.

The *append* command reads the given text and appends it after the addressed line. . is left on the last line input, if there were any, otherwise at the addressed line. Address "**0**" is legal for this command; text is placed at the beginning of the buffer.

(.)**a** *text*

If a space immediately follows the *append* command, then the *text* immediately following the space is appended after the addressed line. . is left at the newly created line. This is essentially a quick method for entering one line.

(. , .)**a**/*text*/

Append the text after the last character in the addressed lines.

b*nn*

The *browse* count is set to *nn* (decimal). This count is then used for subsequent *newline* commands as the number of lines to be printed out. If *nn* is missing, the count is reset to 1.

In constructing addresses as described in rule 11 above, the browse count is added to or subtracted from the current address, instead of a constant of 1 for each + or −. Normally this has no effect since the default is 1.

(. , .)**c**
text
.

The *change* command deletes the addressed lines, then accepts input text which replaces these lines. . is left at the last line input; if there were none, it is left at the first line not deleted.

(. , .)**c**/*regular expression*/*replacement*/
(. , .)**c**/*regular expression*/*replacement*/*nn*
(. , .)**c**/*regular expression*/*replacement*/**g**

This form of the change command is identical to the **s** command below.

(. , .)**co***a*

The **co** (copy) command is identical to the **t** (transfer) command below.

(. , .)**d**

The *delete* command deletes the addressed lines from the buffer. The line originally after the last line deleted becomes the current line; if the lines deleted were originally at the end, the new last line becomes the current line.

d *pathname*
The current directory is set to *pathname* by a call to *chdir*(2).

d=*nn*
Sets *xed*'s idea of what the *depth* of the screen is, to *nn* (decimal) lines. This is used in calculating how many lines will fit on the screen with the : command, and may be preset with the **−c** flag (see above).

e *filename*
ei *filename*
The *edit* command causes the entire contents of the buffer to be deleted, and then the named file to be read in. If no *filename* is given, the *current* file is used. . is set to the last line of the buffer. The number of lines read is printed. *filename* (if present) is remembered for possible use as a default file name in a subsequent **e**, **r**, or **w** command. If the **i** is present, *xed* will read *filename* immediately (without double-checking first).

e=*c*
The *end-of-line* character is set to *c*. Thereafter, any occurrences of *c* are treated as if they were an actual newline character. This facilitates entering several commands on the same physical line. **Caution**: the *eol* character is also interpreted in *insert* mode.

e*nn*
Displays the *long* error message for error number *nn*.

e+
e−
If a − follows, issue error messages in the form ?*nn* where *nn* is the error number of the error that occurred. This is mostly useful for slow terminals. A + returns to long error messages. (See the **−k** flag, and the **e***nn* command above.)

(. , .)**exp**
Providing that a *tab character* has been set (see the **t=***c* command and the **−t** flag) as well as *tab stops* being set (see the **t,***nn* command), any instances of the *tab character* within the addressed lines which are to the left of a column which is marked as a *tab stop*, will be expanded with an appropriate number of *fill characters*. (See the **f=***c* command).

f *filename*
The *filename* command prints the currently remembered file name. If *filename* is given, the currently remembered file name is changed to *filename*.

f=*c*
Set the *fill* character to *c*. This is the character used to fill out a line where *tab* characters have been expanded. If *c* is missing, the *fill* character is reset to the default, which uses as many tabs as possible, followed by as many blanks as necessary to reach the desired column, resulting in the fewest possible characters to get to the desired position.

(**1** , **$**)**g**/*regular expression*/*command-list*
(**1** , **$**)**g**/*regular expression*/**v***command-list*

In the *global* command, the first step is to mark every line which matches the given *regular expression*. If the optional **v** is present after the regular expression, each line potentially matching the regular expression will be printed, followed by the message "**Ok?** ". If the response begins with *n*, the line will not be marked, any other response will cause the line to be marked. Then for every marked line, the given command list is executed with . initially set to that line. A single command or the first of multiple commands appears on the same line with the global command. All lines of a multi-line list except the last line must be ended with \. The **a**, **i**, and **c** commands and associated input are permitted; the . terminating input mode may be omitted if it would be on the last line of the command list. The (global) commands, **g**, and **v**, are not permitted in the command list. If an *end-of-file* (Ctrl-D) is typed in response to the prompt, no further lines will be scanned or marked, and all lines marked so far (if any) will have *command-list* applied to them.

h
h*nn*

Column numbers to column *nn* (default 71) are printed out. Any columns which have *tab* stops set will print out with − character in the appropriate position.

he[lp]

List syntax of all *xed* commands available. (Merely displays the contents of the file **/etc/xed.doc**.)

(.)**i**
text
.

This command inserts the given text before the addressed line. . is left at the last line input; if there were none, at the addressed line. This command differs from the **a** command only in the placement of the text.

(.)**i** *text*

This form of the *insert* command inserts one line before the addressed line, consisting of the *text* following the space. (See the **a** command.)

(. , .)**i**/*text*/

Insert the text before the first character in the addressed lines.

(.−**1** , .)**j**
(.−**1** , .)**j**/*text*/

Join the addressed lines together to form one resulting line. This effectively removes the new-line from the ends of all but the last line. (Useful for rejoining lines that were split incorrectly by the *s* command.)

If a delimiter (and perhaps some *text*) is present, then the *text* will be inserted between the text of the joined lines.

k
(. , .)**k***x*
The mark command marks the addressed line(s) with name *x*, which must be a letter. Either of the address forms ´*x* or *X* (capital letter) then address this/these line(s). If no character is specified after the command, all currently marked lines are listed.

(. , .)**l**
The *list* command prints the addressed lines in an unambiguous way: non-graphic characters are printed as ^*X*, and long lines are folded. *Tab* characters show as > and *backspace* characters are displayed as <. An **l** command may follow most others on the same line.

(.+1 , .+*nn*)**la**
One *page* of text is listed as in the **l** command above. The text is guaranteed not to scroll off the screen.

(1 , $)**ll**
The entire contents of the edit buffer are listed as if "1,$l" had been typed.

m
The characters ^ $. * [& \(\) and \ lose or regain their special meaning in patterns as well as in the substitute command. Each invocation of **m** toggles the "*magic*" characters on/off.

(. , .)**m***a*
(. , .)**mo***a*
The *move* command repositions the addressed lines after the line addressed by *a*. The last of the moved lines becomes the current line.

n
Line numbering is toggled on or off.

n+
n−
Line numbering for the | (and other variants) command is turned on for a +, off for a −.

(. , .)**p**
The *print* command prints the addressed lines. . is left at the last line printed. The **p** command may be placed on the same line after most commands.

(.+1 , .+*nn*)**pa**
One *page* of text is printed out. The text is guaranteed not to scroll off the screen. (See the : command below.)

(1 , $)pp
The entire contents of the edit buffer are listed as if "**1,$p**" had been typed.

q
qi
The *quit* command causes *xed* to exit. No automatic write of a file is done. If the edit file has been modified and the entire contents of the buffer have not been written to a file, a query will be issued to insure that the user has not forgotten to write his file. If the i is present, the editor will quit immediately (without double-checking first). Moreover, if the **−f** flag was selected, the file will *not* be (over)written.

($)r *filename*
The *read* command reads in the given file after the addressed line. If no file name is given, the remembered file name, if any, is used (see *e* and *f* commands). The remembered file name is not changed unless *filename* is the very first file name mentioned. Address "**0**" is legal for *r* and causes the file to be read at the beginning of the buffer. If the read is successful, the number of lines read is typed. . is left at the last line read from the file.

s
The *stop* command without any parameters performs an automatic write (**w**) if the file has been modified and then exits the editor.

(. , .)**s**/*regular expression*/*replacement*/
(. , .)**s**/*regular expression*/*replacement*/*nn*
(. , .)**s**/*regular expression*/*replacement*/**g**
The *substitute* command searches each addressed line for an occurrence of the specified regular expression. On each line in which a match is found, one of the folowing actions are taken for each of the three forms of the command:

1. The first occurrence of the specified expression is replaced by the replacement text.
2. The *nn*-th (where *nn* is a decimal number) occurrence of the specified expression is replaced by the replacement text.
3. All occurrences of the specified expression are replaced.

It is an error for the substitution to fail on all addressed lines. Any character other than *newline* may be used instead of / to delimit the regular expression and the replacement. . is left at the last line substituted.

An ampersand & appearing in the replacement is replaced by the string matching the regular expression. As a more general feature, the characters \n, where *n* is a digit, are replaced by the text matched by the *n*-th regular subexpression enclosed between \(and \). When nested, parenthesized subexpressions are present, *n* is determined by counting occurrences of \(starting from the left.

Lines may be split by substituting *newline* characters into them. The newline in the *replacement* must be escaped by preceding it with a \.

sa*nn*
The *save-count* command changes the default (35) count of text-changing commands

which may be executed before an automatic buffer save will be done. (*nn* is a decimal number.) The save file name is the current filename with a *.edt* extension. A count of zero (0) will disable the auto-save feature.

t

All tab stops currently in effect, as set by the **t,***nn* command, are listed.

(. , .)**t***a*

A copy of the addressed lines is *transferred* after address *a* (which may be 0). . is left at the last line of the copy.

t=*c*

Set *tab* character to *c*. All occurrences of this character entered by the **a** or **i** commands will be expanded to the appropriate number of *fill* characters to get to the next column with a *tab stop*. Any occurrences of the *tab* character after the last tab column will be untouched.

t,*nn,nn,...*

Set *tab stops* in specified (decimal) columns. Numbers preceded by a – will clear the tab setting at that position. The number zero clears *all* tab settings.

u

The *undo* command will restore the last modified line to its original condition. This is different from the **x** (*undelete*) command, which recovers blocks of *deleted* lines, whereas **u** will restore only *one* line, when modified by a *substitution* or *tab* expansion. *Undo* will *not* recover from a *join* command, nor from any deletion, which is processed by the *undelete* command.

(1 , $)**v/***regular expression***/***command-list*
(1 , $)**v/***regular expression***/v***command-list*

This command is the same as the *global* command except that the command list is executed with . initially set to every line **except** those matching the regular expression.

(1 , $)**w** *filename*
(1 , $)**w>***filename*
(1 , $)**wi** *filename*

The *write* command writes the addressed lines onto the given file. If the file does not exist, it is created (see *umask*(2)). The remembered file name is not changed unless *filename* is the very first file name mentioned. If no file name is given, the remembered file name, if any, is used (see **e** and **f** commands). . is unchanged. If the > is present, the addressed lines will be appended onto the end of the file. If the **wi** form is used, and the file is write-locked, then *xed* will attempt to over-ride the file permission, if possible.

w=*nn*

Sets *xed*'s idea of how wide the screen is to *nn* columns. This is used in calculating how many lines will fit on the screen with a : command, and may be preset with the **-w** flag (see above).

(.)x

Undelete is used to recover the most recently deleted (or replaced) block of lines. . is left at the last recovered line.

Example:

25,34d.	delete the lines
*	see the damage
24x	recovers the lost lines

(.)y+
y
y−

This command changes the processing of an interrupt received from the terminal. If the − is present, normal processing takes place. That is, the message "INTERRUPT!" will be displayed on the terminal and *xed* will prompt for another command. If the + is present, the addressed line is set as the initial address for the : command, which will automatically be invoked upon each interrupt. Lastly, if no character follows, then upon each interrupt, one *page* will be displayed from . onward, which is useful for paging through sections of text.

@ *filename*
@p *filename*

Xed opens the specified file, and reads command lines from it. The commands are echoed to the terminal (if the **p** is present) as each character is processed. This allows monitoring the command file as it is running, so that erroneous command line(s) will appear before their respective error messages. If no filename is given, the last *indirect*ed filename, if any, will be used.

!*UNIX-command*

The remainder of the line after the ! is sent to the *shell* (see **SH**(1)) to be interpreted as a *UNIX* command. . is unchanged.

(.)| *UNIX-command*

The addressed lines are *piped* as the standard input to the command(s) following the | symbol. The *UNIX* command is passed to the *shell* (as in ! above) to be processed. Line numbers will not precede the lines of text sent to the command(s) unless explicitly enabled via the **n+** command (see above).

|+
|−

Turn on (or off, respectively) strict checking of the exit status of *UNIX* commands executed via the | | command. If checking is enabled, no processing will be done on the text returned by a command which has a non-zero exit status (thereby implying an error occurred). This reduces the chance of erroneous command processing causing loss of lines. Lines deleted by the | | command may be recovered with **x** (undelete).

(.)| | *UNIX-command*

This variant of the *pipe* command (commonly referred to as the "*double-pipe*" command) performs similarly to the | command above, but replaces the lines sent to the command(s) with those received from the command(s) on the standard output of the command(s). If the error status from the command(s) is not that of a *normal exit*, no change will be made in the text. Similarly, (by default) if the exit status of the command(s) is non-zero (possibly indicating an error) no changes will be made. This is due to the existence of many older programs which do not terminate with a meaningful exit status. The strict exit status checking may be disabled via the |− command below. An optional *line number* (**not** address) may immediately follow the | | which will specify the line after which the returned lines are to be placed.

|<*UNIX-command*

Lines generated by the *UNIX* command(s) are inserted after .. An optional *line number* (**not** address) may immediately follow the < which will specify the line after which the returned lines are to be placed.

(.)|>*UNIX-command*

The only difference between this command and the | | command above is this variant *inserts* the generated text *after* the lines sent, instead of *replacing* the original lines. An optional *line number* (**not** address) may immediately follow the > which will specify the line after which the returned lines are to be placed.

(.+**1** , .+*nn*):
(.-*nn* , .):-
(.-*nn* , .+*nn*)*

One *page* of text is printed out. The text is guaranteed not to scroll off the screen. The first form (just the : alone) will start at the addressed line, the line following . is the default, and print one screenful, or *page* of text. . is set to the last line displayed. The second form, :-, displays one screenful, leaving . as the last line displayed, and remaining as the current line. The last form, *, displays one screenful, with . centered in the *page*.

(.+**1** , .+*nn*)*(newline)*

An address alone on a line causes the addressed line to be printed. A blank line alone is equivalent to "**.+1,.+***nn***p**"; it is useful for stepping through text. The *nn* is the count specified with the **b** command (default 1).

If an interrupt signal (ASCII DEL) is received, *xed* prints "INTERRUPT!" and returns to its command level. (See also the *y* command for alternate interrupt processing.)

Some size limitations

512 characters per line, (see the −**B** flag above)
256 characters per global command list,
64 characters per file name,
128K characters in the temporary file (PDP-11 version only)
(256K characters with −**h** flag)
(No limit on the Vax version)
The limit on the number of lines depends on the amount of core:
each line takes 1 word.
(The current absolute maximum on the PDP-11's is 24,062 lines.)

FILES

/tmp/e?????
: temporary; ????? is process number (in decimal).

/tmp/ep?????
: temporary for | | stuff.

*.hup	if *hangup* signal is received.
*.bak	if **−b** flag is specified.
*.int	if **−i** flag is specified and an *interrupt* is received.
*.edt	auto-save (every 35 commands).
*.trm	if *termination* signal is received.

/etc/xed.doc
: for the **help** command.

DIAGNOSTICS

Each command has self-explanatory error messages.

SEE ALSO

ed(1), edit(1), eed(1), ex(1), umask(2), vi(1)
A Tutorial Introduction to the ED Text Editor – B. W. Kernighan

BUGS

A \ followed by a *newline*, useful for splitting lines with the substitute command, may not be passed through the global command.

If line(s) are deleted which include the endpoints of a range marked with the **k** command, that mark-name character will not work correctly.

B PROGRAMMING LANGUAGE & ENVIRONMENT
CWI

HOW'TO INSTALL Mark1:

Decide on some directory to put the *B* system in, for instance /usr/src/local. Check the tape sticker, and your system's documentation to infer the precise 'tar' command. (Consult the sheet 'The *B* files on ANSI labelled tapes' if you asked for an ANSI tape). Then, mount the tape and type something like:

```
cd /usr/src/local
tar x
```

which will extract all files into the directory /usr/src/local/b. You will need 2.5 megabytes in total to compile and load the system. Now type:

```
cd B
Setup
```

which will ask you some questions to set the *B* system up on your installation. You can call 'Setup' any number of times without spoiling files. So run it once to see what questions you will be asked. If you don't know the answer to some question, you can run it again.

```
make all
```

will compile and load the *B* system, install the 'b' command file in ./bin and the binaries and datafiles it needs in ./lib. You can test the *B* interpreter with:

```
make examples
```

This runs some examples in ./ex. It does not test the *B* editor, however; that can only be done interactively. Try that in ./ex/try. Consult the sheet 'HOW'TO TRY the B editor' (also in ./ex/try/README).

If all is well and you want to make *B* public

```
make install
```

will do some recompilations to get the right pathnames in, and install the 'b' shell command file, the 'b.1' manual file, and the auxiliary files in the directories you indicated during setup. Finally

```
make clean
```

will clean all intermediate object files from the source directories.

To run the *B* system you only need the commands, binaries and data files installed in ./bin and ./lib. If you have made *B* public, all necessary files have been copied to the public places, and you can get rid of the entire *B* file system hierarchy you extracted from the tape, if you want.

If there are any problems, don't panic. Edit the example *Bug Report* form in ./doc to communicate the problem to us. We will then send out diffs for fixed problems in the future.

Above all, we would be *very grateful* to receive any comments you have about the setup procedure, or the *B* system in general, on how to make it easier to use.

Good luck!

HOW'TO TRY the B editor:

The directory B/ex/try is here to try the *B* editor interactively. The example *B* workspace here can always be regenerated with:

```
cd ../generate
cp \'* ../try
cd ../try
```

Now enter the *B* system from this directory with

```
../../bin/b
```

After the *B* system has started up it will prompt for a command with

```
>>> ?
```

Slowly type 's', then 't', (no capitals needed) and you should see the *B* editor suggest the SELECT and START commands, respectively. Now press [TAB] to accept this last suggestion, and [RETURN] to enter the START command to the *B* interpreter. This command will promt you for input, with

```
?
```

Just enter about four lines of text, (which will be echoed), ending with an empty one (press [RETURN] immediately). A short 'poem' should be generated by the *B* interpreter.

If you are already familiar with the *B* language, you might try to edit the START unit by answering

```
>>> ?
```

with

```
:START
```

For example, try to remove the SET'RANDOM command, to get random results on the same input. Or make the unit delay the echoing of the text, entered by the user, until after the reading of the empty line. For testing purposes you should at least try the arrow keys to move the focus around.

You can undo any change by pressing [BACKSPACE].
You can get help with ?.
You can leave the *B* editor with control-X.
You can leave 'b' by typing QUIT.

For more information, see the manual pages 'B(1)' and 'bterminal(5)', and the *User's Guide*.

See the *B* Newsletter, issue 2, for a description of the 'generate' program.

If there are any problems with the editor, consult the 'bterminal(5)' manual entry before trying anything else.

NAME

b – *B* interpreter & environment

SYNOPSIS

b	Starts the *B* interpreter
b –e	Starts the *B* interpreter, using the editor defined in the environment variable EDITOR (*vi* default)
b *file* ...	Makes the *B* interpreter execute the *B* commands in the named file or files. Input for *READ* is taken from standard input.
b –i *table*	An empty permanent table *table* is created; standard input is read, and its lines (considered as texts) are successively put in *table[1], table[2],* ...
b –o *table*	The associates of the permanent table *table* are written to standard output, one to a line.
b –l	The units in the workspace are listed on the standard output.
b –p	The units in the workspace are printed on the system's printer.

DESCRIPTION

A *B* 'work-space' corresponds to a UNIX directory in which the units and permanent targets are kept as separate files.

A call of *b* starts the *B* interpreter; commands can be entered and will be performed immediately. The *B* system takes over control of the screen; it reacts immediately to each key pressed. There is a repertoire of editing operations using function keys and control characters, which may be used to edit the input. A description of all editing operations is given in the User's Guide. The manual entry *bterminal*(5) tells you how the binding of the operations to your terminal's keys can be changed. Here we describe the basic mechanisms using the default key bindings, as listed in the summary at the end.

When the first letter of a command is typed (upper case is not necessary), a possible continuation is *suggested* on the display. The suggestion can be accepted by pressing the [TAB] key; this moves the cursor to the first *hole* (shown as '?') in the suggestion, or to the end of the line if there is none. The suggestion can also be ignored; when more characters are typed the suggestion is changed to conform to these, or removed if nothing applies.

When a command has been completed, it can be executed by pressing [RETURN]. For control commands such as IF and FOR, [RETURN] moves the cursor to an indented position on the next line, awaiting entry of the 'body' of the control command, which may consist of any number of lines. When [RETURN] is pressed twice in succession, this reduces the indentation level; the command is executed when the indentation level is back to zero.

Corrections can be made by pressing the [BACKSPACE] key. This cancels the effect of any key pressed, including [TAB] and the editing operations, as well as [RETURN] within a control command. Repetition of [BACKSPACE] cancels more keys, to a maximum of 100 (currently). Once execution of a command has started, it cannot be corrected, though it can be stopped by pressing [BREAK] at any time. If the copy-buffer is empty (see the Copy command), the last command executed, last input provided or last text deleted is saved in the buffer, and can be retrieved with [control-C]. It may then be edited using standard editing operations.

When the user types the first line of a HOW'TO-, YIELD- or TEST-command, editing of the unit continues in a similar way to editing an IF or FOR command. The user can complete the unit through standard editing operations, and finish by pressing [control-X] or several [RETURN]s. Further commands can then be given.

An existing unit can be edited by typing, on the command level, a colon (:) followed by the unit name. Similarly, an existing target can be edited by typing an equals-sign (=) followed by the target name. The name may be left out in subsequent edit requests for the same object, or for the unit that most recently caused an error message.

When a unit that is being edited gets longer than the screen size, a scroll bar will be displayed at the bottom of the screen. It shows approximately which part of the unit is visible on the screen. If your terminal has the Goto operation, and you *goto* some place on the scroll bar, the *B* system will reposition the visible part of the unit accordingly.

A double colon (::) given at the command level lists the headings of the units in the present workspace. Likewise, a double equals-sign (==) gives the names of the permanent targets.

A call of **b -e** starts the *B* interpreter, but uses the editor defined by the environment variable EDITOR. If this is not set then *vi* is used. The editor is then only used for entering units, or editing existing units; immediate commands cannot be edited.

The other calls of *b* do not activate the interpreter, but provide communication between the workspace and UNIX, as described above in the synopsis.

FILES

$HOME/.Bed_buf	copy buffer (if locked) between sessions
$HOME/.Bed_pos	focus position of last 50 edited units
.b_perm	table mapping object names to file names
.b_temp	scratch file
.Bed_sugg	suggestion list for user-defined commands
'*, <*, "*, >*	units in this workspace
=*	permanent targets in this workspace

SEE ALSO

bterminal(5)

Lambert Meertens, *Draft Proposal for the B Programming Language; Semi-formal Definition,* Mathematical Centre, 1982.

Lambert Meertens and Steven Pemberton, *Description of B,* CWI (formerly Mathematical Centre), 1984. Also: SIGPLAN Notices, Vol. **20**, No.2, February 1985.

Steven Pemberton, *A User's Guide to the B System,* CWI, 1984.

B Quick Reference card. Available from CWI.

Leo Geurts, *Computer Programming for Beginners, Introducing the B Language; Part 1,* CWI, 1984. (Also available in Dutch.)

Leo Geurts, *An Overview of the B Programming Language, or B without Tears,* SIGPLAN Notices, Vol. **17**, No. 12, December 1982.

AUTHORS

Frank van Dijk, Leo Geurts, Timo Krijnen, Lambert Meertens, Steven Pemberton, Guido van Rossum

SUMMARY OF EDITING OPERATIONS

Name	Default Keys†	Short description
Accept	[TAB]	Accept suggestion, focus to hole or end of line
Return	[RETURN]	Add line or decrease indentation
Widen	*f1*, [ESC] w	Widen focus
First	*f2*, [ESC] f	Move focus to first contained item
Last	*f3*, [ESC] l	Move focus to last contained item
Extend	*f4*, [ESC] e	Extend focus (usually to the right)
Upline	*f5*, [ESC] u	Move focus to whole line above
Previous	*f6*, [ESC] p	Move focus to previous item
Next	*f7*, [ESC] n	Move focus to next item
Downline	*f8*, [ESC] d	Move focus to whole line below
Up	↑, [ESC] U	Make new hole, move up
Down	↓, [ESC] D	Make new hole, move down
Left	←, [ESC] ,	Make new hole, move left
Right	→, [ESC] .	Make new hole, move right
Goto	[ctrl-G]	New focus at cursor position
Undo	[BACKSPACE]	Undo effect of last key pressed (may be repeated)
Redo	[ctrl-U]	Redo last UNDOne key (may be repeated)
Delete	[ctrl-D]	Delete contents of focus (to buffer if empty)
Copy	[ctrl-C]	Copy buffer to hole, or focus to buffer
Record	[ctrl-R]	Start/stop recording keystrokes
Play	[ctrl-P]	Play back recorded keystrokes
Look	[ctrl-L]	Redisplay screen
Help	[ESC]?	Print summary of editing operations
Exit	[ctrl-X]	Finish changes or execute command
Interrupt	[BREAK], [DEL]	Interrupt command execution

† Notes:

The binding of editing operations to keys may be different for your terminal; see *bterminal*(5) for more information.

Keys named *f1*...*f8* are *function keys.* The way to type these is terminal-dependent. The codes they send must be defined by the termcap entry for your terminal. See *bterminal*(5).

If a terminal has arrow keys ↑, ←, →, ↓ which transmit codes to the computer, these should be used for Up, Down, Left and Right. Again, the termcap entry must define the codes.
The Goto operation can only be used if the cursor can be moved locally at the terminal; the Goto operation will sense the terminal for the cursor position, using two extra non-standard termcap capabilities; see *bterminal*(5) for more details.

If you have set your interrupt character with *stty*(1) to something other than [DEL], you can type [ctrl-]] for Interrupt.

[Ctrl-D] means: hold the [CTRL] (or [CONTROL]) key down while pressing d.

[ESC] w means: press the [ESC] key first, then w.

NAME

bterminal – adapting the *B* system to your local terminals

DESCRIPTION

The *B* system uses the termcap library to address the terminal, and determines the codes sent by your terminal's function keys from the termcap database. To this end it uses the environment variables TERM and TERMCAP to determine the type and capabilities of your terminal. (See *tset*(1) and *termcap*(5) for the exact use of termcap.)

You can also redefine the binding of editing operations in a *key definitions file.* There are a number of places where this file can be found, so that there can be different key bindings per terminal and per user. It is even possible to define an environment variable giving this place.

DEFAULT KEY BINDINGS

The following table gives the names of the editing operations, and the default bindings.

Name	Default bindings	Termcap bindings
accept	^I (1)	
return	^M	
widen	"\ew"	*k1* (2)
first	"\ef"	*k2*
last	"\el"	*k3*
extend	"\ee"	*k4*
upline	"\eu"	*k5*
previous	"\ep"	*k6*
next	"\en"	*k7*
downline	"\ed"	*k8*
up	"\eU"	*ku*
down	"\eD"	*kd*
left	"\e,"	*kl*
right	"\e."	*kr*
goto	^G	
undo	^H	
redo	^U	
delete	^D	
copy	^C	
record	^R	
play	^P	
look	^L	
help	"\e?"	
exit	^X	
ignore (3)		
term_init (4)		*ks*
term_done (4)		*ke*

Notes:

(1) ^X means the Control-X character, \e means escape; see below for an exact description of the format of key definitions.

(2) The termcap entries *k1* ... *k8* describe the codes sent by the function keys, and *ku* ... *kd* decribe the codes sent by the arrow keys.

(3) With the name *ignore* you can declare input strings illegal; see below.

(4) The termcap entries *ks* and *ke* are sent to the terminal at startup and upon exiting.

The third column of the table describes additional bindings for some operations that are derived from termcap, if the termcap entry for your terminal defines that capability. If a termcap definition conflicts with some other default, the definition derived from the termcap holds. (For instance, on a Televideo the left arrow key sends ^H; this means that the binding of *undo* to ^H (or [BACKSPACE]) is no longer valid.)

KEY DEFINITIONS FILE

Each line in the key definitions file contains one definition or a comment. A definition consists of the name of the editing operation (see the table above), an equals sign (=), and one or more *items*. Each *item* can be a string, a number, or a control-character. The latter is written as '^' followed by a letter. A number is an octal number if it starts with 0 (it should not include 8 or 9, then), otherwise it is decimal; it stands for the corresponding ASCII character. Strings are delimited by single (') or double (") quotes. Inside strings, the following escape sequences are recognized:

\ddd	(one to three octal digits) the ASCII character ddd
\b	backspace, 010
\e	escape, 033
\f	formfeed, 014
\n	linefeed, 012
\r	carriage return, 015
\t	tab, 011
char	any other character (notably \ or ' or ")

Finally, everything from a '#' character to the end of the line is taken as a comment (except inside a string, of course).

Each definition implies that the concatenation of its items, when received as input, will provoke the execution of the designated editing operation. A definition for the 'operation' *ignore* means that this input string will be treated as an illegal operation (the *B* system will try to ring the bell).

Definitions for the *term_init* and *term_done* operations define strings that will be sent *to* the terminal at initialization time, and on exiting, respectively. These can be used to set programmable function keys, for instance.

Note that the definitions in the file only *add* to the already existing key bindings (see the defaults above). When one definition is an initial subsequence of the other, the last one given in the file holds. It is probably counter-productive to have the first item of a definition start with a printable character, as this would make it impossible to enter that character.

Beware that you cannot use ^] as a key binding when your interrupt character is not the default [DEL] key; see below.

Examples:

```
widen = "\ep\r"       # HP-2621 function key 1 is escape p return
accept = ^I           # tab
undo = ^A "O" ^M      # unshifted function key 11 on Televideo
                      # used because left arrow overwrites ^H
ignore = 0177         # ignore DEL (a common noise character)
```

The first of the following four files found by the B system is used to read key bindings from:

```
$HOME/.Bed_$TERM
B_LIB/.Bed_$TERM
$HOME/.Bed_def
B_LIB/.Bed_def
```

Here $HOME and $TERM are the values of the environment variables; see sh(1) and tset(1). B_LIB stands for a directory appointed by your system administrator (normally /usr/new/lib/B) where various auxiliary files for the *B* system are kept. This organization allows different key bindings per user and per terminal.

HELP FILE

The file B_LIB/Bed_help contains a screenful of help information, describing the editing operations and the keys to which they are bound. If you change the key bindings this information is not correct anymore, and so you can define an environment variable BED_HELP, that gives the pathname of the file to be printed when the *help* operation is executed.

INTERRUPT

To interrupt the execution of a *B* command you should normally use the [BREAK] key. If your interrupt character is the [DEL] key, this will also work. However, if you have set your interrupt character with *stty*(1) to something other than [DEL], the *B* system will in turn reset it to ^]. This is done to prevent a collision with a key that accesses one of the *B* editing operations, like ^C for copy. It means that you cannot use ^] as a key binding in a description file in this case.

INVERSE VIDEO (standout mode)

If your terminal skips a position on the screen when switching to or from inverse video, you are out of luck. The *B* editor must be able to display part of a word in inverse video and the rest normal, without surrounding spaces in between. You can still use *B* (without the *B* editor) with the command **b -e**; see *b*(1).

GOTO OPERATION

The Goto operation can only be used on terminals that can move the cursor locally, eg. the arrow keys do not send any codes to the host computer. If your terminal can be *sensed* for the cursor position, then you can use the operation to tell it you moved the cursor away. You should ask your system administrator to add the non standard capabilities *sp* and *cp* to the termcap entry for your terminal. The *sp* capability should define the string sent by the *B* system to the terminal to request the cursor position. The *cp* capability must define the format of the cursor position string as returned by the terminal; most of the % escapes as defined in termcap(5) for cursor addressing are recognized. (For example,

```
cp=\E&a%r%3c%3Y^M:sp=\E`\021
```

are the entries for a HP2621 terminal.)

FILES

B_LIB	/usr/new/lib/B, unless changed by your system administrator
$HOME/.Bed_$TERM	key definitions file; first of these four holds
B_LIB/.Bed_$TERM	
$HOME/.Bed_def	
B_LIB/.Bed_def	
$BED_HELP	file with one screenful of help info
B_LIB/Bed_help	default help file

BUGS

In searching for the key definitions file .Bed_$TERM the *B* system doesn't recognize aliases for terminal types. Watch out for variations like e.g. TERM=2621-wl.

```
=================================================================
=          The B programming language and environment.          =
=================================================================
```

Authors: Frank van Dijk
                Leo Geurts
                Timo Krijnen
                Lambert Meertens
                Steven Pemberton
                Guido van Rossum

                Centrum voor Wiskunde en Informatica
                Department of Computer Science
                POB 4079
                1900 AB Amsterdam
                The Netherlands

Net address: ...{decvax,philabs}!mcvax!timo

Description:

B is a new programming language and environment for personal computing being designed and implemented at the CWI.
We have tried to combine attractive features in existing systems with some ideas of our own.

Some of the good points of B programming language proper are:
* a powerful collection of only five different data types that may easily be combined;
* strong typing, yet without declarations;
* no limitations, apart from sheer exhaustion of memory;
* refinements to support top-down programming;
* nesting by indentation.

Some of the good points of the B environment are:
* no need for files; units (procedures and functions) and global variables remain after logging out;
* one consistent face is shown to the user at all times, whether she executes commands, edits units, or enters input to a program;
* generalized undo mechanism.

The Mark 1 distribution is a full implementation of the language, with a small environment that includes a B dedicated editor front-end to the interpreter, variables that survive logging out, and independently editable program units.

```
# Copyright (c) Stichting Mathematisch Centrum, Amsterdam, 1984.

=====================================================================
= The Structure of the file system of the Mark 1 Implementation of B. =
=====================================================================

#
# DIRECTORY STRUCTURE
#

bin     place to install 'b' shell command file within B file system.

ex      example B workspaces.

doc     documentation.

lib     place to install auxiliary files and binaries within B file system.

man     manuals.

src     sources for the B system.

src/b
        sources for 'b' shell command file and its auxiliary files.
src/bed
        sources for the B editor 'bed'.
src/bint
        sources for the B interpreter 'bint'.
src/libbed
        sources for auxiliary files needed by the B editor.
src/libtermcap
        sources for the termcap library needed by the B editor.

#
# README's and MAKEFILE's
#

All source directories have README and Makefile files.
These Makefiles accept the following entry points:

make install    build new version of a program or library and install it
                together with auxiliary files in ./bin or ./lib.

make clean      remove unnecessary files that can easily be rebuilt.

make print      print sources, documentation or manuals.

#
# generic files
#

Most source Makefile's and src/b/b.sh and src/b/b_p.sh are generated
from generic copies by ./Setup. If you need to edit any of these and
still be able to run ./Setup later, you must edit the xx.gen generic
version of the file, and install it by running ./Setup.
```

Bug Report

FROM

Name:

Firm/Institute:

Address:

Telephone:

Internet network address:

Machine Type:	O vax	O sun	O pdp	O other: ....
Operating System:	O 4.2 BSD	O Version 7	O System V	O other: ....

==
B VERSION: Mark1C.
==

FAULTY PROGRAM:

DESCRIPTION OF FAULT:

REPEAT BY:

==

Send to

B Group
Informatics / AA
CWI
POB 4079
1009 AB Amsterdam
The Netherlands

or by electronic mail to

...{decvax,philabs,seismo}!mcvax!timo

DIPRESS - XEROX INTERPRESS TOOLS
Xerox

NAME

charset – print all characters within a given character subset of an interpress font

SYNOPSIS

charset [**–p***subset*] [**–s***pointsize*] [**–o***outputfile*] fontname

DESCRIPTION

Charset will create an interpress master which will print a specified character subset of an interpress font at a given point size.

The specification for fontname is treated as the final identifier of the universal naming scheme in §3.2.2 in *Interpress Electronic Printing Standard* (XSIS 048404). The name specified is appended to "Xerox/XC1-1-1/" to form the full interpress name.

The following options are understood:

–p*subset*
: The number of the character subset to be printed. This is specified in octal and defaults to character subset 0.

–s*pointsize*
: The point size to be used for the font to be printed. This is specified in decimal and defaults to 10.

–o*outputfile*
: The name of the file to be created as the interpress master. If the option is omitted, the interpress master will be created on the standard ouptut, so it is best to redirect this to a file or pipe to the queueing software. Note that *outputfile* may directly follow the **–o** or it may be the next argument in the list.

FILES

/usr/local/lib/font/devipress device and font information

SEE ALSO

Interpress Electronic Printing Standard, XSIS 048404

AUTHOR

William LeFebvre, Webster Research Center, Xerox Corporation

BUGS

The output file is not automatically queued for printing.

NAME

dipress – convert device independent troff output to interpress

SYNOPSIS

dipress [**–f***directory*] [**–o***list*] [**–t**] [file] ...

DESCRIPTION

Dipress take the output of device independent *troff*(1) found in the files specified, produce an interpress file, and queue that file for printing on the interpress printer. If no files are specified, then standard input is used. The following options are understood:

–f*directory*
: Take font information from *directory* instead of the default.

–o*list*
: Print pages whose numbers are given in the comma-separated *list*. The list contains single numbers *N* and ranges *N1–N2*. A missing *N1* means the lowest numbered page, a missing *N2* means the highest.

–t
: Direct output to standard output rather than the printer. No queueing is performed if this option is specified. Redirecting standard output is highly recommended when this option is used.

FILES

/usr/local/lib/font/devipress device and font information

SEE ALSO

troff(1)

AUTHOR

William LeFebvre, Webster Research Center, Xerox Corporation

BUGS

If you have Services 11 or earlier on your printer, long boxes will experience a small overshoot at the right end.

For a similar reason, it is recommended that the **–D** option be used with *pic(1)*. Any pictures built with the drawing commands will not show up on the Xerox 8700/9700. That printer uses a different representation for bitmaps than the 8044.

NAME

ipmetrics – convert an Interpress metrics master to other forms

SYNOPSIS

ipmetrics [**–t**] [[**–T**] [**–d** destinationLibrary] file ...

DESCRIPTION

Ipmetrics executes the Interpress metric masters given on the command line and converts the results to the metrics formats for various composition systems. Currently the system will produce metrics for Troff and TeX.

The default option is **–t** which means to produce metrics files for troff. The **–T** says to produce metrics for TeX. The **–d** option specifies the final destination of the output metrics files. This program doesn't place the metrics files there immediately, it uses this option to help generate a shell script to do the installation. The default for this option is site dependent.

On the standard input is a description of the mapping from the font names in the metric master to target font names. The standard–input is token–based where tokens are seperated by white space. A "#" in column one (1) indicates a comment. The first token on the standard input must be "device" followed by the name of the target device. Each line after that contains the description of one font. Each of these lines has five (5) tokens in it. The first three are the full universal name of the font, the next is the name as it's known to the destination composition system and the last token names a file that will contain the mapping specification for the font. That file will specify how to translate from the metric master's character set (usually XC1–1–1) to that of the destination system. Note that a font in the metrics master can be named multiple times to create several logical fonts for one physical font. Example:

```
# this is a commment
device 8044
Xerox XC1-1-1 TroffClassic       R   troffClassic.map
Xerox XC1-1-1 TroffMathExtra         RN  romanNumerials.map
Xerox XC1-1-1 TroffMathExtra         XX  xerox-eXtra.map
```

The *.map files are used to map between XC1–1–1 and the destination system. Each line should specify a different character in the 16 bit character space. The first token specifies the high–order eight bits in octal and the second token the low–order eight bits (again in octal). The third byte specifies the Troff ascender/descender information and the rest of the line specifies various names for that character. Example:

```
#char (a|de)sender    alias
0 041  2        !
0 042  2        "
```

$LIB/fonts/* fonts *.map mapping files

SEE ALSO

The Interpress Toolkit manual, dipress(1), troff(1)

BUGS

Fuzzyness about the nature of what the DVI -> Interpress converter wants make the TeX mode less than perfect. Fuzzyness about the "easy" property of a master makes handling sizes non–uniform.

NAME

iptotext – convert an interpress file into intertext

SYNOPSIS

iptotext [**–d**] [**–o** *outputfile*] [file ...]

DESCRIPTION

Iptotext will convert an interpress file into a readable Ascii form. This textual form, called *intertext*, consists of, basically, numbers and operator names. The output can be edited by any conventional text editor and converted back to an interpress file with *texttoip*(1).

The **–d** option will cause pixel arrays to be formatted and included in the output. The default is to not dump pixel array data because of its voluminous nature.

If **–o** is specified, the text is written to the file *outputfile*. If no output file name is given, the output appears on standard out. Note that this is a little different than *texttoip*(1).

AUTHOR

William LeFebvre

SEE ALSO

texttoip(1)

Intertext–a Textual Representation of Interpress, William LeFebvre
Interpress Electronic Printing Standard, XSIS 048404

NAME

iptroff – convert troff to Interpress

SYNOPSIS

iptroff [**-D** device] [**-o** output-file] [**-t**] troff arguments

DESCRIPTION

Itroff runs *troff*(1) sending its output through *dipress*(1) to produce typeset output for an Interpress printer. As the document is processed, the names of the passes will be printed. This can be suppressed with the **-q** (quiet) switch. If the troff **-t** switch is specified, however, the troff output will be sent directly to the standard output and not piped through *dipress*. In addition, the **-q** option is assumed with the **-t** option. If the **-D** switch is specified, then the font metrics for the named device will be used instead of the default set. The actual printer that the Interpress master gets sent to is site dependent. If the **-o** option is specified, then the Interpress master is left in the specified file rather than being transmitted to a print server.

The default font is TroffClassic which is the normal Xerox Classic font augmented to provide all the special characters found in the troff manual with the single exception of the Bell System Logo.

Example:

```
iptroff -ms paper
```

will set a paper with the "ms" macro package in the TroffClassic.

FILES

$LIB/fonts/* default font mounts and bug fixes

SEE ALSO

The Interpress Toolkit manual, dipress(1), troff(1)

BUGS

The TroffClassic font is the only font currently being exported.

NAME

maha – make and print interpress files

SYNOPSIS

maha [options] [files]

DESCRIPTION

Maha (maharani–the Interpress version of the program czarina) reads in text files, converts them to interpress format and ships them to an interpress printer. It also performs some simple page formatting.

The environment variable MAHA may be used to specify default options. The value of MAHA is parsed as a string of arguments before the arguments that appear on the command line. For example, "MAHA='–f Classic/8'" sets your default body font to 8 point Classic.

The possible options are:

–*n* Prints output *n* columns per page (note that *n* is limited to one digit)

–b *banner*
: Uses *banner* to label the output. It will appear on the cover page on the line labeled "Document".

–c *n* Causes *n* copies of the output to be printed. The default is 1.

–f *font*
: Sets the font to be used for the body of each page. The default is "Vintage–Printwheel/10".

–F *font* Sets the font to be used for page headings. The default is "Modern–Bold/12".

–H *header*
: Sets the format for page headings to the string *header*. Certain formatting options can be embedded in this string. See the section below entitled "Header Format". The default header is constructed from the file name, its last modification date, and a page and line number.

–h *header*
: Appends the string *header* to the current header format string. This can be used to append something to the default header.

–l Causes line printer simulation mode to be used: pages will be 66 lines long and headers will be omitted.

–n *name*
: Sets the delivery address of your output (the "For" field on the cover sheet) to *name*. The default is your full name as recorded in the gecos field of the password file ("/etc/passwd").

–o *file*
: The interpress code is written into *file*. The default is generated from the process i.d. of the program.

–r Rotates the output 90 degrees on the page (landscape mode). This is good for output that requires a wide paeg or for program listings when specifying two columns. Some people like the program listings produced by the command "**maha –2 –r** files".

–R Forces portrait mode. This overrides the **–r** option. It is useful if the environment variable MAHA sets **–r**.

–s *pages*
: Selects pages to be printed. *Pages* may be a single page specification (eg. "5"), a range of pages ("5–10"), or a list of page specifications (eg. "3,11–13"). Note that this syntax is identical to that accepted by the **–o** option of *troff*.

-t Causes page headings (titles) to be omitted.

Specifying Fonts

The naming scheme for fonts is a slight variant on the universal naming scheme used by interpress. See §3.2.2 in *Interpress Electronic Printing Standard* (XSIS 048404) for a description of universal names. Each identifier of a universal font name is separated by a slash ("/"). If the final identifier in the name is nothing more than a series of digits, as in "Classic/8", then it is taken to be a point size. The actual universal name is formed by removing this last identifier. If the last part of the name is not strictly a number, then the point size is assumed to be 10. Since most environments use the same prefix for a universal font name, A standard prefix of several identifiers is prepended to every font name given on the command line. This can be overridden by placing a slash at the front of the given name in which case the universal name is formed by simply removing the slash from the front of the name. Note that the point size calculation mentioned above will still be performed on this type of name.

Here are some examples that should clear the air:

Name Given	Universal Name	Point Size
Classic	Xerox/XC1-1-1/Classic	10
Modern/12	Xerox/XC1-1-1/Modern	12
/Rice/TimesRoman	Rice/TimesRoman	10
/Rice/TimesRoman/Italic/8	Rice/TimesRoman/Italic	8
/Rice/VileBlob/8/10	Rice/VileBlob/8	10

If a font name has the unfortunate characteristic of containing nothing but numbers in its last identifier, it can still be specified by always appending a point size to the name, as in the last example above.

Header Format

The string that is used to build the header can have format options embedded in it. Each formatting option is preceded with the character "%" in a manner similar to *printf*(3S) strings in C. The following format characters are recognized:

f	current file name
t	last modified time of the current file
p	current page number
l	line number for the top line of the current page

If a percent sign is followed by a character not in the list above, then that character gets printed (without the leading percent sign). Note that a percent sign can still be printed in the header by placing two in the format. The default format string is:

```
"%f          %t          Page %p, line %l"
```

ENVIRONMENT

MAHA strings of options to be used by *maha*.

FILES

/etc/passwd contains information about system users

SEE ALSO

cz(1) czarina

Interpress Electronic Printing Standard, XSIS 048404

BUGS

The document name doesn't appear on the banner page like it should. This is really the fault of the queueing software.

Maha will not realize that a file is already in interpress format and skip the conversion phase.

NAME

texttoip – convert an intertext file into interpress

SYNOPSIS

texttoip [**–o** *outputfile*] [file ...]

DESCRIPTION

Texttoip will convert a textual representation of an interpress file (called an *intertext* file) into an actual interpress file. The utility *iptotext*(1) generates output that is suitable to be used as input to this program. If **–o** is specified, the interpress codes are written to the file *outputfile*. If no output file name is given, the output is written to the file "intertext.ip" unless output has been redirected away from the terminal. Sending interpress codes directly to a terminal is not considered to be a worthwhile operation. If more than one file is specified as input, the interpress header is taken from the first input file and any header specifications in subsequent files are ignored.

AUTHOR

William LeFebvre

SEE ALSO

iptotext(1)

Intertext–a Textual Representation of Interpress, William LeFebvre
Interpress Electronic Printing Standard, XSIS 048404

ICON SYSTEM
Arizona

An Overview of the Icon Programming Language*

Ralph E. Griswold

TR 83-3a

May 13, 1983

Department of Computer Science

The University of Arizona

Tucson, Arizona 85721

*This work was supported by the National Science Foundation under Grant MCS81-01916.

An Overview of the Icon Programming Language

1. Introduction

Icon is a high-level programming language with extensive facilities for processing strings and lists. Icon has several novel features, including expressions that may produce sequences of results, goal-directed evaluation that automatically searches for a successful result, and string scanning that allows operations on strings to be formulated at a high conceptual level.

Icon resembles SNOBOL4 [1] in its emphasis on high-level string processing and a design philosophy that allows ease of programming and short, concise programs. Like SNOBOL4, storage allocation and garbage collection are automatic in Icon, and there are few restrictions on the sizes of objects. Strings, lists, and other structures are created during program execution and their size does not need to be known when a program is written. Values are converted to expected types automatically; for example, numeral strings read in as input can be used in numerical computations without explicit conversion. Whereas SNOBOL4 has a pattern-matching facility that is separate from the rest of the language, string scanning is integrated with the rest of the language facilities in Icon. Unlike SNOBOL4, Icon has an expression-based syntax with reserved words; in appearance, Icon programs resemble those of several other conventional programming languages.

Examples of the kinds of problems for which Icon is well suited are:

- text analysis, editing, and reformatting
- document preparation
- symbolic mathematics
- text generation
- program parsing and translation
- data laundry
- graph manipulation

Icon is implemented in C [2] and runs under UNIX* on the PDP-11, VAX-11, and Onyx C8002 computers. Implementations for other computers and operating systems are presently underway. An earlier version of Icon [3] is available on several large-scale computers, including the CRAY-1, DEC-10, IBM 360/370, PRIME 450/550/650, DG MV8000, and CDC Cyber/6000.

A brief description of some of the representative features of Icon is given in the following sections. This description is not rigorous and does not include many features of Icon. See [4] for a complete description.

2. Strings

Strings of characters may be arbitrarily long, limited only by the architecture of the computer on which Icon is implemented. A string may be specified literally by enclosing it in double quotation marks, as in

```
greeting := "Hello world"
```

which assigns an 11-character string to **greeting**, and

*UNIX is a trademark of Bell Laboratories.

```
address := ""
```

which assigns the zero-length *empty* string to **address**. The number of characters in a string **s**, its size, is given by ***s**. For example, ***greeting** is 11 and ***address** is 0.

Icon uses the ASCII character set, extended to 256 characters. There are escape conventions, similar to those of C, for representing characters that cannot be keyboarded.

Strings also can be read in and written out, as in

```
line := read()
```

and

```
write(line)
```

Strings can be constructed by concatenation, as in

```
element := "(" || read() || ")"
```

If the concatenation of a number of strings is to be written out, the **write** function can be used with several arguments to avoid actual concatenation:

```
write("(",read(),")")
```

Substrings can be formed by subscripting strings with range specifications that indicate, by position, the desired range of characters. For example,

```
middle := line[10:20]
```

assigns to **middle** the string of characters of **line** between positions 10 and 20. Similarly,

```
write(line[2])
```

writes the second character of **line**. The value 0 is used to refer to the position after the last character of a string. Thus

```
write(line[2:0])
```

writes the substring of **line** from the second character to the end, thus omitting the first character.

An assignment can be made to the substring of string-valued variable to change its value. For example,

```
line[2] := "..."
```

replaces the second character of **line** by three dots. Note that the size of **line** changes automatically.

There are many functions for analyzing strings. An example is

```
find(s1,s2)
```

which produces the position in **s2** at which **s1** occurs as a substring. For example, if the value of **greeting** is as given earlier,

```
find("or",greeting)
```

produces the value 8. See Section 4.2 for the handling of situations in which **s1** does not occur in **s2**, or in which it occurs at several different positions.

3. Character Sets

While strings are sequences of characters, *csets* are sets of characters in which membership rather than order is significant. Csets are represented literally using single enclosing quotation marks, as in

```
vowels := 'aeiouAEIOU'
```

Two useful built-in csets are **&lcase** and **&ucase**, which consist of the lowercase and uppercase letters, respectively. Set operations are provided for csets. For example,

```
letters := &lcase ++ &ucase
```

forms the cset union of the lowercase and uppercase letters and assigns the resulting cset to **letters**, while

```
consonants := letters -- 'aeiouAEIOU'
```

forms the cset difference of the letters and the vowels and assigns the resulting cset to **consonants**.

Csets are useful in situations in which any one of a number of characters is significant. An example is the string analysis function

```
upto(c,s)
```

which produces the position s at which any character in c occurs. For example,

```
upto(vowels,greeting)
```

produces 2. Another string analysis function that uses csets is

```
many(c,s)
```

which produces the position in s after an initial substring consisting only of characters that occur in s. An example of the use of **many** is in locating words. Suppose, for example, that a word is defined to consist of a string of letters. The expression

```
write(line[1:many(letters,line)])
```

writes a word at the beginning of **line**. Note the use of the position returned by a string analysis function to specify the end of a substring.

4. Expression Evaluation

4.1 Conditional Expressions

In Icon there are *conditional expressions* that may *succeed* and produce a result, or may *fail* and not produce any result. An example is the comparison operation

```
i > j
```

which succeeds (and produces the value of **j**) provided that the value of **i** is greater than the value of **j**, but fails otherwise.

The success or failure of conditional operations is used instead of Boolean values to drive control structures in Icon. An example is

```
if i > j then k := i else k := j
```

which assigns the value of **i** to **k** if the value of **i** is greater than the value of **j**, but assigns the value of **j** to **k** otherwise.

The usefulness of the concepts of success and failure is illustrated by **find(s1,s2)**, which fails if s1 does not occur as a substring of s2. Thus

```
if i := find("or",line) then write(i)
```

writes the position at which **or** occurs in **line**, if it occurs, but does not write a value if it does not occur.

Many expressions in Icon are conditional. An example is **read()**, which produces the next line from the input file, but fails when the end of the file is reached. The following expression is typical of programming in Icon and illustrates the integration of conditional expressions and conventional control structures:

```
while line := read() do
   write(line)
```

This expression copies the input file to the output file.

If an argument of a function fails, the function is not called, and the function call fails as well. This "inheritance" of failure allows the concise formulation of many programming tasks. Omitting the optional **do** clause in **while-do**, the previous expression can be rewritten as

```
while  write(read())
```

4.2 Generators

In some situations, an expression may be capable of producing more than one result. Consider

```
sentence := "Store it in the neighboring harbor"
find("or",sentence)
```

Here **or** occurs in **sentence** at positions 3, 23, and 33. Most programming languages treat this situation by selecting one of the positions, such as the first, as the result of the expression. In Icon, such an expression is a *generator* and is capable of producing all three positions.

The results that a generator produces depend on context. In a situation where only one result is needed, the first is produced, as in

```
i := find("or",sentence)
```

which assigns the value 3 to **i**.

If the result produced by a generator does not lead to the success of an enclosing expression, however, the generator is *resumed* to produce another value. An example is

```
if (i := find("or",sentence)) > 5 then write(i)
```

Here the first result produced by the generator, 3, is assigned to **i**, but this value is not greater than 5 and the comparison operation fails. At this point, the generator is resumed and produces the second position, 23, which is greater than 5. The comparison operation then succeeds and the value 23 is written. Because of the inheritance of failure and the fact that comparison operations return the value of their right argument, this expression can be written in the following more compact form:

```
write(5 < find("or",sentence))
```

Goal-directed evaluation is inherent in the expression evaluation mechanism of Icon and can be used in arbitrarily complicated situations. For example,

```
find("or",sentence1) = find("and",sentence2)
```

succeeds if **or** occurs in **sentence1** at the same position as **and** occurs in **sentence2**.

A generator can be resumed repeatedly to produce all its results by using the **every-do** control structure. An example is

```
every i := find("or",sentence)
   do write(i)
```

which writes all the positions at which **or** occurs in **sentence**. For the example above, these are 3, 23, and 33.

Generation is inherited like failure, and this expression can be written more concisely by omitting the optional **do** clause:

```
every write(find("or",sentence))
```

There are several built-in generators in Icon. One of the most frequently used of these is

```
i to j
```

which generates the integers from **i** to **j**. This generator can be combined with **every-do** to formulate the traditional **for**-style control structure:

```
every k := i to j do
   f(k)
```

Note that this expression can be written more compactly as

```
every f(i to j)
```

There are a number of other control structures related to generation. One is *alternation*,

$expr_1$ | $expr_2$

which generates the results of $expr_1$ followed by the results of $expr_2$. Thus

```
every write(find("or",sentence1) | find("or",sentence2))
```

writes the positions of **or** in **sentence1** followed by the positions of **or** in **sentence2**. Again, this sentence can be written more compactly by using alternation in the second argument of **find**:

```
every write(find("or",sentence1 | sentence2))
```

Another use of alternation is illustrated by

```
(i | j | k) = (0 | 1)
```

which succeeds if any of **i**, **j**, or **k** has the value 0 or 1.

5. String Scanning

The string analysis and synthesis operations described in Sections 2 and 3 work best for relatively simple operations on strings. For complicated operations, the bookkeeping involved in keeping track of positions in strings becomes burdensome and error prone. In such cases, Icon has a string scanning facility that is analogous in many respects to pattern matching in SNOBOL4. In string scanning, positions are managed automatically and attention is focused on a current position in a string as it is examined by a sequence of operations.

The string scanning operation has the form

s ? *expr*

where **s** is the *subject* string to be examined and *expr* is an expression that performs the examination. A position in the subject, which starts at 1, is the focus of examination.

Matching functions change this position. One matching function, **move(i)**, moves the position by **i** and produces the substring of the subject between the previous and new positions. If the position cannot be moved by the specified amount (because the subject is not long enough), **move(i)** fails. A simple example is

```
line ? while write(move(2))
```

which writes successive two-character substrings of **line**, stopping when there are no more characters.

Another matching function is **tab(i)**, which sets the position in the subject to **i** and also returns the substring of the subject between the previous and new positions. For example,

```
line ? if tab(10) then write(tab(0))
```

first sets the position in the subject to 10 and then to the end of the subject, writing **line[10:0]**. Note that no value is written if the subject is not long enough.

String analysis functions such as **find** can be used in string scanning. In this context, the string that they operate on is not specified and is taken to be the subject. For example,

```
line ? while write(tab(find("or")))
   do move(2)
```

writes all the substrings of **line** prior to occurrences of **or**. Note that **find** produces a position, which is then used by **tab** to change the position and produce the desired substring. The **move(2)** skips the **or** that is found.

Another example of the use of string analysis functions in scanning is

```
line ? while tab(upto(letters)) do
   write(tab(many(letters)))
```

which writes all the words in **line**.

As illustrated in the examples above, any expression may occur in the scanning expression. Unlike SNOBOL4, in which the operations that are allowed in pattern matching are limited and idiosyncratic, string scanning is completely integrated with the rest of the operation repertoire of Icon.

6. Structures

6.1 Lists

While strings are sequences of characters, lists in Icon are sequences of values of arbitrary types. Lists are created by enclosing the lists of values in brackets. An example is

```
carl := ["buick","skylark",1978,2450]
```

in which the list **carl** has four values, two of which are strings and two of which are integers. Note that the values in a list need not all be of the same type. In fact, any kind of value can occur in a list – even another list, as in

```
inventory := [carl,car2,car3,car4]
```

Lists also can be created by

```
a := list(i,x)
```

which creates a list of **i** values, each of which has the value **x**.

The values in a list can be referenced by position much like the characters in a string. Thus

```
carl[4] := 2400
```

changes the last value in **carl** to 2400. A reference that is out of the range of the list fails. For example,

```
write(carl[5])
```

fails.

The values in a list **a** are generated by **!a**. Thus

```
every write(!a)
```

writes all the values in **a**.

Lists can be manipulated like stacks and queues. The function **push(a,x)** adds the value of **x** to the left end of the list **a**, automatically increasing the size of **a** by one. Similarly, **pop(a)** removes the leftmost value from **a**, automatically decreasing the size of **a** by one, and produces the removed value.

A list value in Icon is a pointer (reference) to a structure. Assignment of a structure in Icon does not copy the structure itself but only the pointer to it. Thus the result of

```
demo := carl
```

causes **demo** and **carl** to reference the same list. Graphs with loops can be constructed in this way. For example,

```
nodel := ["a"]
node2 := [nodel,"b"]
push(nodel,node2)
```

constructs a structure that can be pictured as follows:

```
node1  . ->a - - .
       |        |
       |        |
node2  ' - -b<- '
```

6.2 Tables

Icon has a table data type similar to that of SNOBOL4. Tables essentially are sets of pairs of values, an *entry value* and a corresponding *assigned value*. The entry and assigned values may be of any type, and the assigned value for any entry value can be looked up automatically. Thus tables provide a form of associative access in contrast with the positional access to values in lists.

A table is created by an expression such as

```
symbols := table(x)
```

which assigns to **symbols** a table with the default assigned value **x**. Subsequently, **symbols** can be referenced by any entry value, such as

```
symbols["there"] := 1
```

which assigns the value 1 to the **there**th entry in symbols.

Tables grow automatically as new entry values are added. For example, the following program segment produces a table containing a count of the words that appear in the input file:

```
words := table(0)
while line := read() do
   line ? while tab(upto(letters)) do
      words[tab(many(letters))] +:= 1
```

Here the default assigned value for each word is 0, as given in **table(0)**, and **+:=** is an augmented assignment operation that increments the assigned values by one. There are augmented assignment operations for all binary operators.

Tables can be converted to lists, so that their entry and assigned values can be accessed by position. This is done by **sort(t)**, which produces a list of two-element lists from **t**, where each two-element list consists of an entry value and its corresponding assigned value. For example,

```
wordlist := sort(words)
every pair := !wordlist do
   write(pair[1]," : ",pair[2])
```

writes the words and their counts from **words**.

7. Procedures

An Icon program consists of a sequence of procedure declarations. An example of a procedure declaration is

```
procedure max(i,j)
   if i > j then return i else return j
end
```

where the name of the procedure is **max** and its formal parameters are **i** and **j**. The **return** expressions return the value of **i** or **j**, whichever is larger.

Procedures are called like built-in functions. Thus

```
k := max(*s1,*s2)
```

assigns to **k** the size of the longer of the strings s1 and s2.

A procedure also may suspend instead of returning. In this case, a result is produced as in the case of a return, but the procedure can be resumed to produce other results. An example is the following procedure that generates the words in the input file.

```
procedure genword()
   local line, letters, words
   letters := &lcase ++ &ucase
   while line := read() do
      line ? while tab(upto(letters)) do {
         word := tab(many(letters))
         suspend word
         }
end
```

The braces enclose a compound expression.

Such a generator is used in the same way that a built-in generator is used. For example

```
every word := genword() do
   if find("or",word) then write(word)
```

writes only those words that contain the substring **or**.

8. An Example

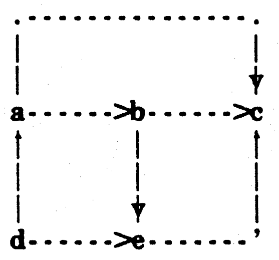
The following program sorts graphs topologically.

```
procedure main()
   local sorted, nodes, arcs, roots
   while nodes := read() do {                    # get next node list
      arcs := read()                             # get arc list
      sorted := ""                               # sorted nodes
                                                 # get nodes without predecessors
      while *(roots := nodes -- snodes(arcs)) > 0 do {
         sorted ||:= roots                       # add to sorted nodes
         nodes --:= roots                        # delete these nodes
         arcs := delarcs(arcs,roots)             # delete their arcs
         }
      if *arcs = 0 then write(sorted)            # successfully sorted
      else write("graph has cycle")              # cycle if node remains
   }
end
```

```
procedure snodes(arcs)
   local nodes
   nodes := ""
   arcs ? while move(1) do {                    # predecessor
      move(2)                                   # skip "->"
      nodes ||:= move(1)                        # successor
      move(1)                                   # skip ";"
      }
   return nodes
end

procedure delarcs(arcs,roots)
   local newarcs, node
   newarcs := ""
   arcs ? while node := move(1) do {            # get predecessor node
      if many(roots,node) then move(4)          # delete arc from root node
      else newarcs ||:= node || move(4)         # else keep arc
      }
   return newarcs
end
```

Graph nodes are represented by single characters with a list of the nodes on one input line followed by a list of arcs. For example, the graph



is given as

```
abcde
a->b;a->c;b->c;b->e;d->a;d->e;e->c;
```

for which the output is

```
dabec
```

The nodes are represented by csets and automatic type conversion is used to convert strings to csets and vice versa. Note the use of augmented assignment operations for concatenation and in the computation of cset differences.

Acknowledgement

Icon was designed by the the author in collaboration with Dave Hanson, Tim Korb, Cary Coutant, and Steve Wampler. The current implementation is largely the work of Cary Coutant and Steve Wampler with recent contributions by Bill Mitchell. Dave Hanson and Bill Mitchell also made several helpful suggestions on the presentation of material in this paper.

References

1. Griswold, Ralph E., Poage, James F., and Polonsky, Ivan P. *The SNOBOL4 Programming Language*, second edition. Prentice-Hall, Inc., Englewood Cliffs, New Jersey. 1971.
2. Kernighan, Brian W. and Ritchie, Dennis M. *The C Programming Language*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey. 1978.
3. Griswold, Ralph E. *Differences Between Versions 2 and 5 of Icon*, Technical Report TR 83-5, Department of Computer Science, The University of Arizona. 1983.
4. Griswold, Ralph E. and Griswold, Madge T. *The Icon Programming Language*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey. 1983.

Extensions to Version 5 of the Icon Programming Language*

Ralph E. Griswold

Robert K. McConeghy

William H. Mitchell

TR 84-10a

June 27, 1984; Revised August 4, 1984

Department of Computer Science

The University of Arizona

Tucson, Arizona 85721

*This work was supported by the National Science Foundation under Grants MCS81-01916 and DCR-8401831.

Extensions to Version 5 of the Icon Programming Language

1. Introduction

The standard features of Version 5 of Icon are described in Reference 1. Since Icon is the byproduct of a research effort that is concerned with the development of novel programming language facilities for processing nonnumeric data, it is inevitable that some extensions to the standard language will develop.

Some of these extensions are incorporated as features of new releases. Others are available as options that can be selected when the Icon system is installed [2]. This report describes the extensions that are included in Version 5.9 of Icon.

All the extensions are upward-compatible with standard Version 5 Icon. Their inclusion should not interfere with any program that works properly under the standard version.

2. New Version 5.9 Features

2.1 The Link Directive

Version 5.9 contains a link directive that simplifies the inclusion of separately translated libraries of Icon procedures. If *icont(1)* [3] is run with the **−c** option, source files are translated into intermediate *ucode* files (with names ending in **.u1** and **.u2**). For example,

```
icont −c libe.icn
```

produces the ucode files **libe.u1** and **libe.u2**. The ucode files can be incorporated in another program with the new link directive, which has the form

```
link libe
```

The argument of **link** is, in general, a list of identifiers or string literals that specify the names of files to be linked (without the **.u1** or **.u2**). Thus

```
link libe, "/usr/icon/ilib/collate"
```

specifies the linking of **libe** in the current directory and **collate** in **/usr/icon/ilib**.

The environment variable *IPATH* controls the location of files specified in link directives. *IPATH* should be have a value of the form *p1:p2: ... pn* where each *pi* names a directory. Each directory is searched in turn to locate files named in link directives. The default value of *IPATH* is '.', that is, the current directory.

2.2 Installation Options

When an Icon system is installed, various configuration options are specified [2]. The value of the keyword **&options** is a string that contains the command line arguments that were used to configure Icon.

3. Optional Extensions

There are two extension options: sets (**−sets** in **&options**), and a collection of experimental features (**−xpx** in **&options**).

3.1 Sets

Sets are unordered collections of values and have the properties normally associated with sets in the mathematical sense. The function

```
set(a)
```

creates a set that contains the distinct elements of the list a. For example,

```
set(["abc",3])
```

creates a set with two members, abc and 3. Note that

```
set([])
```

creates an empty set. Sets, like other data aggregates in Icon, need not be homogeneous — a set may contain members of different types.

Sets, like other Icon data aggregates, are represented by pointers to the actual data. Sets can be members of sets, as in

```
s1 := set([1,2,3])
s2 := set([s1,[]])
```

in which s2 contains two members, one of which is a set of three members and the other of which is an empty list.

Any specific value can occur only once in a set. For example,

```
set([1,2,3,3,1])
```

creates a set with the three members 1, 2, and 3. Set membership is determined the same way the equivalence of values is determined in the operation

```
x === y
```

For example,

```
set([[],[]])
```

creates a set that contains two distinct empty lists.

The functions and operations of Icon that apply to other data aggregates apply to sets as well. For example, if s is a set,

```
*s
```

is the size of s (the number of members in it). Similarly,

```
type(s)
```

produces the string set and

```
s := set(["abc",3])
write(image(s))
```

writes set(2). Note that the string images of sets are in the same style as for other aggregates, with the size enclosed in parentheses.

The operation

```
!s
```

generates the members of s, but in no predictable order. Similarly,

```
?s
```

produces a randomly selected member of s. These operations produce values, not variables — it is not possible to assign a value to !s or ?s.

The function

```
copy(s)
```

produces a new set, distinct from s, but which contains the same members as s. The copy is made in the same fashion as the copy of a list — the members themselves are not copied.

The function

```
sort(s)
```

produces a list containing the members of s in sorted order. Sets themselves occur after tables but before records in the sorting order.

The customary set operations are provided. The function

```
member(s,x)
```

succeeds and returns the value of x if x is a member of s, but fails otherwise. Note that

```
member(s1,member(s2,x))
```

succeeds if x is a member of both s1 and s2.

The function

```
insert(s,x)
```

inserts x into the set s and returns the value of s (it is similar to **put(a,x)** in form). Note that

```
insert(s,s)
```

adds s as an member of itself.

The function

```
delete(s,x)
```

deletes the member x from the set s and returns the value of s.

The functions **insert(s,x)** and **delete(s,x)** always succeed, whether or not x is in s. This allows their use in loops in which failure may occur for other reasons. For example,

```
s := set([])
while insert(s,read())
```

builds a set that consists of the (distinct) lines from the standard input file.

The operations

```
s1 ++ s2
s1 ** s2
s1 -- s2
```

create the union, intersection, and difference of s1 and s2, respectively. In each case, the result is a new set.

The use of these operations on csets is unchanged. There is no automatic type conversion between csets and sets; the result of the operation depends on the types of the arguments. For example,

```
'aeiou' ++ 'abcde'
```

produces the cset **abcdeiou**, while

```
set([1,2,3]) ++ set([2,3,4])
```

produces a set that contains 1, 2, 3, and 4. On the other hand,

```
set([1,2,3]) ++ 4
```

results in Run-time Error 119 (**set expected**).

Examples

Word Counting:

The following program lists, in alphabetical order, all the different words that occur in the standard input file:

```
procedure main()
   letter := &lcase ++ &ucase
   words := set([])
   while text := read() do
      text ? while tab(upto(letter)) do
         insert(words,tab(many(letter)))
   every write(!sort(words))
end
```

The Sieve of Eratosthenes:

The follow program produces prime numbers, using the classical "Sieve of Eratosthenes":

```
procedure main(a)
   local limit, s, i
   limit := a[1] | 5000                          # limit to 5000 if not specified
   s := set([])
   every insert(s,1 to limit)
   every member(s,i := 2 to limit) do
      every delete(s,i + i to limit by i)
   primes := sort(s)
   write("There are ",*primes," primes in the first ",limit," integers.")
   write("The primes are:")
   every write(right(!primes,*limit + 1))
end
```

4. Expermental Features

4.1 PDCO Invocation Syntax

The experimental features include the procedure invocation syntax that is used for programmer-defined control operations [4]. In this syntax, when braces are used in place of parentheses to enclose an argument list, the arguments are passed as a list of co-expressions. That is,

p{$expr_1$**,** $expr_2$**, ...,** $expr_n$**}**

is equivalent to

p([create $expr_1$**, create** $expr_2$**, ..., create** $expr_n$**])**

Note that

p{}

is equivalent to

p([])

4.2 Invocation Via String Name

The experimental features allow a string-valued expression that corresponds to the name of a procedure or operation to be used in place of the procedure or operation in an invocation expression. For example,

```
"image"(x)
```

produces the same call as

```
image(x)
```

and

```
"-"(i,j)
```

is equivalent to

```
i - j
```

In the case of operations, the number of arguments determines the operation. Thus

```
"-"(i)
```

is equivalent to

```
-i
```

Since **to-by** is an operation, despite its reserved-word syntax, it is included in this facility with the string name.... Thus

```
"..."(1,10,2)
```

is equivalent to

```
1 to 10 by 2
```

Similarly, range specifications are represented by ":", so that

```
":"(s,i,j)
```

is equivalent to

```
s[i:j]
```

Defaults are not provided for omitted or null-valued arguments in this facility. Consequently,

```
"..."(1,10)
```

results in a run-time error when it is evaluated.

The subscripting operation also is available with the string name []. Thus

```
"[]"(&lcase,3)
```

produces **c**.

String names are available for all the operations in Icon, but not for control structures. Thus

"|"(*expr*$_1$,*expr*$_2$)

is erroneous. Note that string scanning is a control structure.

Field references, of the form

expr . *fieldname*

are not operations in the ordinary sense and are not available via string invocation.

String names for procedures are available through global identifiers. Note that the names of functions, such as **image**, are global identifiers. Similarly, any procedure-valued global identifier may be used as the string name of a procedure. Thus in

```
global q

procedure main()
   q := p
   "q"("hi")
end

procedure p(s)
   write(s)
end
```

the procedure **p** is invoked via the global identifier **q**.

4.3 Conversion to Procedure

The experimental features include the function **proc(x,i)**, which converts **x** to a procedure, if possible. If **x** is procedure-valued, its value is returned unchanged. If the value of **x** is a string that corresponds to the name of a procedure as described in the preceding section, the corresponding procedure value is returned. The value of **i** is used to distinguish between unary and binary operators. For example, **proc("^",2)** produces the exponentiation operator, while **proc("^",1)** produces the co-expression refresh operator. If **x** cannot be converted to a procedure, **proc(x,i)** fails.

4.4 Integer Sequences

To facilitate the generation of integer sequences that have no limit, the experimental features include the function **seq(i,j)**. This function has the result sequence {i, i+j, i+2j, ... }. Omitted or null values for i and j default to 1. Thus the result sequence for **seq()** is {1, 2, 3, ... }.

Acknowledgements

The design of sets for Icon was done as part of a class project. In addition to the authors of this paper, the following persons participated in the design: John Bolding, Owen Fonorow, Roger Hayes, Tom Hicks, Robert Kohout, Mark Langley, Susan Moore, Maylee Noah, Janalee O'Bagy, Gregg Townsend, and Alan Wendt.

References

1. Griswold, Ralph E. and Madge T. Griswold. *The Icon Programming Language*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey. 1983.

2. Griswold, Ralph E. and William H. Mitchell. *Installation and Maintenance Instructions for Version 5.9 of Icon*, Technical Report TR 84-13, Department of Computer Science, The University of Arizona. August 1984.

3. Griswold, Ralph E. and William H. Mitchell. *ICONT(1)*, manual page for *UNIX Programmer's Manual*, Department of Computer Science, The University of Arizona. August 1984.

4. Griswold, Ralph E. and Michael Novak. "Programmer-Defined Control Operations", *The Computer Journal*, Vol. 26, No. 2 (May 1983). pp. 175-183.

Version 5.9 of Icon

Ralph E. Griswold, Robert K. McConeghy, and William H. Mitchell

August 22, 1984

Version 5.9 of Icon is a modification of Version 5.8 of Icon. The UNIX* implementation runs on both PDP-11s and VAXs. This document is a brief summary of Version 5.9. See also References 1, 2, and 3.

Changes

- An optional language extension provides sets as a built-in data type [3].
- The Icon compiler has been deleted, leaving only the interpreter. A "personalized interpreter" facility has been added to allow individuals to maintain customized versions of the Icon run-time system [5]. This facility replaces the former use of external functions with the Icon compiler to augment the function repertoire of Icon.
- The implementation of the table data type has been redone to increase the efficiency of table lookup.
- There is a new keyword, **&options**, whose value is a string listing the options used for installing Icon at the local site [3, 4].
- Considerable work has been done throughout the Icon system to improve the quality of the code and to remove nonportable constructs.
- The source code has been commented extensively.
- A number of minor bugs have been fixed.
- The Icon program library has beed reorganized and new material has been added to it [6].
- The Icon distribution hierarchy has been reorganized and new material has been added to aid in testing and porting to new computers [4].

User Impacts

- Persons who formerly used the Icon compiler with external functions will need to convert to personalized interpreters.
- The internal organization of tables is different from earlier implementations. For example, if t is a table, the order of elements generated by !t generally is different from before. Similarly, the value of ?t is likely to be different.
- The functionality of some components of the Icon program library has been changed.

UNIX* is a trademark of AT&T Bell Laboratories.

Known Bugs

This list ennumerates all known bugs in Version 5.8 of Icon. If you find a bug that is not in this list, please contact us.

- The translator does not detect arithmetic overflow in conversion of numeric literals. Very large numeric literals may have incorrect values.
- Integer overflow on multiplication and exponentiation are not detected during execution. This may occur during type conversion.
- Line numbers may be wrong in diagnostic messages related to lines with continued quoted literals.
- In some cases, trace messages may show the return of subscripted values, such as **&null[2]**, that would be erroneous if they were dereferenced.
- File names are truncated to 14 characters by some versions of UNIX. If such a truncation deletes part of the terminating **.icn** of a file that is input to the translator, mysterious diagnostic messages may occur during linking.
- On PDP-11s, list blocks can contain no more than 4090 elements. List blocks are created when the **list()** function is called, when literal lists are specified, and when the **sort()** function converts a table into a list. It should be noted that it is possible for a list to grow to beyond 4090 elements; the limitation is only upon the size of the list when it is created.
- There is a bug in the 4.1bsd **fopen()** routine that under certain conditions returns a **FILE** pointer that is out of range when one tries to open too many files. On systems where this bug is present, it may manifest itself in the form of run-time Error 304 when one tries to open too many files. (On 4.1bsd systems this limit is usually 20 files.)
- If one has an expression such as x := **create** ... in a loop, and x is not a global variable, the unreferenceable expression stacks generated by each successive **create** operation are not garbage collected. This problem can be circumvented by making x a global variable or by assigning a value to x before the **create** operation, e.g., x := **&null**; x := **create** ....
- Overflow of a co-expression stack due to excessive recursion is not detected and may cause mysterious program malfunction.
- Program malfunction may occur if **display()** is used in a co-expression.
- The garbage collector was designed for machines with small address spaces and as such is not well-suited for machines like the VAX. No empirical studies have been made, but it is suspected that performance of the garbage collector could be improved substantially on the VAX. In particular, if the user attempts to create a very large data object that will not fit into memory, (such as a million-element list), it takes the system an inordinately long time to determine that the object can not be allocated.

References

1. Griswold, Ralph E. *An Overview of the Icon Programming Language*. Technical Report TR 83-3a, Department of Computer Science, The University of Arizona. May 1983.

2. Griswold, Ralph E. and Madge T. Griswold. *The Icon Programming Language*. Prentice-Hall Inc., Englewood Cliffs, New Jersey. 1983.

3. Griswold, Ralph E., Robert K. McConeghy, and William H. Mitchell. *Extensions to Version 5 of the Icon Programming Language*. Technical Report TR 84-10a, Department of Computer Science, The University of Arizona. August 1984.

4. Griswold, Ralph E. and William H. Mitchell. *Installation and Maintenance Guide for Version 5.9 of Icon*, Technical Report TR 84-13, Department of Computer Science, The University of Arizona. August 1984.

5. Griswold, Ralph E., Robert K. McConeghy, and William H. Mitchell. *Personalized Interpreters for Icon*. Technical Report TR 84-14, Department of Computer Science, The University of Arizona. August 1984.

6. Griswold, Ralph E. *The Icon Program Library*. Technical Report TR 84-12, Department of Computer Science, The University of Arizona. August 1984.

MMDF MAIL SYSTEM
Dr. Dave Farber, Delaware

MMDFII: A Technical Review

Douglas P. Kingston III

Ballistic Research Laboratory
Aberdeen Proving Grounds, Maryland 21005
<dpk@brl>

ABSTRACT

The Multi-channel Memo Distribution Facility (MMDF) is one of the most sophisticated mail systems available for the UNIX† operating system. MMDF is a mail transport system that supports a variety of user interfaces and delivery mechanisms. The design was not encumbered with the need to be compatible with existing mail systems, and as a result MMDF has a unified family of mail handling programs. This review will discuss MMDF's design and operation, concentrating on those features that are unique to MMDFII, the latest release of MMDF.

MMDF's design allows it to grow from a single-host system to a large mail relay without degradation of mail system performance, and to degrade gracefully as the load becomes huge. The demands of a high volume mail relay have led to many of MMDF's innovative design choices.

Unlike some other systems, MMDF has separate processes for mail submission and delivery. Recent changes to the delivery software to permit intelligent retry strategies based on the retry history for each dead host will be explained. The effect of the new domain server mechanism on address validation will be discussed.

The separation of mail into channels is key to MMDF's ability to handle large amounts of mail. Each channel represents a different class of delivery and each channel has its own queue. This isolates problems and allows one to provide different "levels of service" to different channels.

Other topics to be discussed will include available user interfaces, the mailing list processor, aliasing, runtime configuration, and domain based naming.

The MMDF system was originally developed at the University of Delaware and has since seen significant development work at the Ballistic Research Laboratory and University College London.

Introduction and History

The Multi-channel Memo Distribution Facility, commonly called MMDF, is a suite of software that has seen a great deal of work since it was originally released in 1980. The original code was designed and implemented by Dave Crocker working under Professor David Farber at the University of Delaware (UDEL). The MMDF system was then chosen to form the initial backbone software for the CSNET project and has been in use for several years by elements of the U.S. Army. The software has seen a great deal of change in the process. The original code is commonly referred to as MMDFI or MMDF Version 1. A number of minor additions and changes were made while fielding MMDFI as the result of collaboration between UDEL and BRL and some other sites. After the original code was fielded in CSNET, Dave Crocker began the development of a upgraded version of the MMDF

† Unix is a trademark of Bell Laboratories.

system which was designed to work in the new Internet domain naming system and was to incorporate numerous design changes suggested by experience with MMDFI. Dave Crocker left the CSNET project before completing this work, approximately two weeks before the TCP/IP switchover of the ARPANET, 1 January 1983. At this time, BRL was a solid MMDF site. We were reluctant to try to retrofit the existing version of MMDFI to handle the new mail protocols that also took effect on 1 January, so Doug Kingston of BRL undertook the task of finishing the work needed to make MMDFII operational. A production version of MMDFII was installed at BRL during the third week of January 1983, and served as BRL's mail system on three hosts, but there was no stable version of the MMDFII code until June 1983. The first few months of MMDFII were quite rough and it needed a great deal of "tender loving care".

For reasons that will be clear in a moment, this stable version of June 1983 is now referred to as the MMDFII-pre-England version. Around June, a copy of this stable version was delivered to Steve Kille of University College London (UCL) and to Brendan Reilly of UDEL, who had taken over Dave Crocker's work on MMDF at UDEL. Steve Kille made a number of major changes to the handling of domains, address parsing, and handling of the alias files. Steve also added support for NIFTP, a European file transfer protocol used for sending mail in a batch environment. At the same time that Steve was making his enhancements, Doug Kingston continued to develop BRL's copy of MMDFII to make it an even more solid mail system. BRL's changes were not as major as Steve's but covered a great deal of code and fixed several major outstanding bugs. This dual development led to two variants of MMDFII that each needed the other's improvements. In late September of 1983 Brendan Reilly and Doug Kingston spent a week in England with Steve to merge the variants and to discuss future changes and directions for MMDF. The result of this meeting was a merged version of MMDFII which I will call MMDFII-post-England. Just prior to this trip, the CSNET Information Center (CIC) received a copy of the pre-England MMDF. Their later changes were based on this pre-England version which made merging of their changes into the post-England version somewhat difficult.

After the England meeting, Brendan Reilly of UDEL took the role of coordinator of the subsequent changes to MMDF. Copies of the MMDF-post-England were made simultaneously available to BRL, UCL, and UDEL. Since then many minor changes have been made by all four sites; in essentially all cases these changes have been bug fixes or changes to make MMDF a more stable and robust system.

Since then, Doug Kingston at BRL has made changes to the local delivery mechanism, rewriting much of the original code, and the central delivery program has been upgraded to take advantage of large-address-space machines, when possible, to keep retry histories for messages on a host-by-host basis. Bernie Cosell at the CIC has undertaken to speed up MMDF execution by providing a facility for compiling in some of the information normally included in the ASCII text-based version. Steve Kille an alternative to the ASCII text based version. Steve Kille has continued to refine the address handling and the British "backwards" domain code. [1] Brendan Reilly has made changes to the package to allow it to run on the Altos system and has fixed numerous bugs in the PhoneNet code.

The MMDF Design

The MMDF system design has not changed fundamentally from the original design proposed and implemented for MMDFI. The design of MMDFI is covered in detail in the paper "An Internetwork Memo Distribution Capability". [2] In this chapter, I will summarize the basic design and

1 The British do domains backwards. For example, if in the US (Internet) we write "user@VAX1.EE.UDEL.ARPA" known as "little endian" order, the British (SERC Net) write "user@ARPA.UDEL.EE.VAX1" or "big endian" order. Put another way, "big endians" put the largest, most general, or most significant element of the domain first. "Little endians" use the other order, with the most significant part last. [See *Gulliver's Travels* by Joanthan Swift. The "big endian" vs. "little endian" controversy was a *causus belli* in Lilliput.]

2 D. Crocker, E. Szurkowski, and D. Farber, "An Internetwork Memo Distribution Capability", Datacom Conference, September 1979.

discuss those changes that have been made in MMDFII.

Mail software is normally classed into one of two groups. A user agent (UA) is a program that is responsible for providing a "user friendly" interface for reading or writing mail and converting to the canonical interface of the mail transfer agent as necessary. A mail transfer agent (MTA) is a program or system to which you or your user agent entrusts a message for delivery to someone else's user agent. There has been a great deal of confusion caused by people who fail to realize the difference between a mail transfer agent and a user agent, or even that a difference exists! There is a wide variety of user agents which can be used with MMDF, and it is the responsibility of the use agent to provide a user interface. This separation of the functions of user agent and mail transfer agent has many advantages, not the least of which is that MMDF can support many different user interfaces with ease. Currently, there are at least five different interfaces available including the Rand MH system, V6 style mail, Berkeley's Mail (cap-mail), the Tenex-style Send and Msg programs, and Rmail (for UUCP). MMDF is a mail transfer agent. MMDF does not have, nor does it claim to have, a good "user interface"; instead it has a good program (MTA-UA) interface. MMDF accepts messages for delivery either locally or to a remote site. It attempts to verify the validity of the addresses at submission time to the extent possible given only a host table and a list of local addresses. If accepted, it will continually try to resend the message until the retry time is exhausted at which time the message is returned to the sender.

The MMDF system can be thought of as two subsystems, responsible for mail submission and mail delivery, respectively. Between these two halves is the mail queue. The mail queue will be discussed later, but basically it stores each message as two files, an address list with some control information, and a separate file containing only the message text (header and body).

The submission half of the system consists mainly of one program, called "**submit**", which is responsible for enqueuing mail to be delivered. As much verification as possible is performed on the message at submission time. For mail destined for the local machine, this means making sure the destination account exists, and that any local mailing list or aliases expand properly. For mail that has a non-local host specification, the **submit** process checks to see if it knows how to reach the specified host. Mail which for any reason is known to be undeliverable, is not accepted for delivery. **Submit** is called by two types of processes. The first group includes user agents such as the **send** program. The second group comprises channel programs such as **rmail** [3] which are interfacing to remote mail transfer agents.

The delivery portion of MMDF is represented by two main elements: the **deliver** program which manages the queue, and the channel programs which handle the details of delivery to a specific network, host, or mail system. The **deliver** program takes each message which is eligible to be delivered, and opens the appropriate address list. For each address in the list, **deliver** ensures that it is running the appropriate channel program and then passes the envelope information [4] to the channel. A reference to the file containing the actual message text is passed to the channel program. The channel decides how to deliver the message and sees to any necessary message reformatting that may be necessary (e.g. "header munging").

There is currently a variety of channels and the number is growing. The local channel handles delivery of messages to local addresses. The list channel is a special, somewhat incestuous, channel which acts the role of channel, receiving user agent, and sending user agent all in one. The list channel resubmits mail back into the mail system by calling **submit**. This has several benefits that will be discussed later. [5] The SMTP channel delivers mail via TCP/IP connections using the SMTP protocol. [6] The phone channel uses the PhoneNet link-level protocol software developed at the University of Delaware for sending mail over dialup or hard-wired terminal lines. The NIFTP channel queues

3 The **rmail** program (/bin/rmail) is invoked by uucp when delivering mail on your system.

4 The MMDF envelope information consists of the the return address, the destination addresses, some delivery options and a reference to the message text file.

5 See the section on the list channel.

6 J. Postel, "RFC821 - Simple Mail Transfer Protocol", Network Information Center, SRI International, August 1982.

mail as files to be transferred using the NIFTP protocol used in the British research community. The UUCP channel is used to queue requests for transfer using UUCP.

Development of MMDFII was started about the same time that the Sendmail mail transfer agent was being written by Eric Allman at Berkeley. Dave Crocker met with Eric on a number of occasions and was impressed by his work. Some elements of the Berkeley software were so useful that they inspired the development of similar facilities in MMDFII. Not the least of these was the runtime configuration file. It is now possible to configure the MMDF software totally from a single, ASCII-text-based configuration file. Unlike Sendmail's terse configuration syntax, MMDF uses much more verbose keyword/value pairing for configuration information. MMDFII can be configured either from compiled-in values, for fast startup, or totally from the text configuration file, or from any combination of the two. As a result of the fact that many values can be compiled-in, the usual MMDF tailoring file is one-tenth the size of a Sendmail tailoring file, and can be even smaller, even on a large relay site. The runtime configuration file is one of the most useful additions in MMDFII, especially for sites supporting more than one host, since one can now run the same binaries on all machines of the same type. Another Berkeley-inspired facility was the ability to have an alias file entry that forces a delivery to a file or pipe. While initially insecure, this facility has been made reliable by adding code to /fBsubmit/fR that knows when you are processing an alias file entry and when you are processing some other type of address. Simple ownership of the file containing the address was not considered sufficient protection since there are too many files left writable to the world that are owned by root or other privileged users.

The MMDF File Hierarchy

The MMDFI queue structure consisted of three directories. The "msg" directory contained the files containing actual message text. The "addr" directory contained the address list file for each message, and the "tmp" directory contained the address list while it was being created before moving it to the "addr" directory. The names of the files in "msg" and "addr" were identical although the contents differed. This made it easy to find the companion file given either filename. The MMDFII queue structure has changed in one major way from that of MMDFI. There is now one extra directory per channel named q.*channel.* If an address list still has any addresses destined to be sent on any channels, then a link will exist from the address list file in "addr" to a file with the same name in each queue directory which is referenced. All the above directories usually live in /usr/mmdf/lock/home, although the root name of this tree can be changed. The lock directory is kept in 700 or 770 mode, accessible only to the "mmdf" user and possibly a "systems" group for ease of maintenance. The "home" directory and all the underlying queue directories are kept in 777 mode to allow ease of movement and access for the trusted but unprivileged programs that operate there. In particular, the **submit** process is setuid to "mmdf" to allow it to enter the tree, but it then restores its UID and GID to those of the invoker. **Deliver** and the channel programs normally run as the "mmdf" user. Only the local channel, which must access the real user's mailbox files, and the TCP/IP network daemons, which must access privileged sockets, need run privileged. There are several other programs that are setuid to "root", but only so they can change their UID to "mmdf" so as to to be a "trusted submitter", which allows them to specify an arbitrary From: line.

The addition of the separate queuing directories on a per channel basis was a valuable change. UNIX performs poorly with large directories, as any UUCP backbone site can tell you. This new mechanism allows easy partitioning of channel activities into separate directories, since **deliver** will never access the copy of the address list in the "addr" directory until it goes to finally expunge the message from the queues. If one channel or site gets backed up, it does not affect the performance of any of the other channels.

The format of the address lists has changed somewhat since MMDFI. The old format was:

S T channel-name host-on-channel address-at-host

where S was either a '–' indicating unsent or a '+' indicating that only the address but not the text had been sent. The T was either the type of delivery (almost always 'm' for mail) or a '*' indicating the message has been completely delivered. The channel-name and host-on-channel should be obvious. The address-at-host parameter was only the local part of an address. The new version differs in two ways. First, the host-on-channel is now the full domain address of the host to deliver to, as specified in the routing information of the domain table. [7] Second, the address-at-host is now the full address with both the local-part and the domain specifier.

In the future, I will probably change the location of the "message completely delivered" flag to be in the first position (S), since I feel it was a mistake to not have it there in the past and it is confusing in its current position. This has not been done as of February 1985.

The MMDF file hierarchy also has a logging directory. Separate logs are kept here for **submit** and **deliver**, the channel programs, and the phone dialing package. This directory is generally accessible although unreadable and the logs are normally write only. This could be made "mmdf" access only for some sites, with the only penalty being that some programs would be unable to add entries to the log. A subdirectory of the log directory called "log/phase" contains time stamp files whose modification times are changed by **submit** and **deliver** to indicate such things as last pickup time, last delivery time, last poll made, and similar information.

There are two other directories in the MMDF hierarchy which should be mentioned. The "chans" directory contains all of the channel programs invoked by the **deliver** program, and other ancillary daemon programs such as the SMTP daemon and the SMTP server. The "table" directory contains all files necessary to maintain the MMDF database, including domain tables, host address files, mailbox alias files, dialing scripts, and programs to build these files and incorporate them into the DBM library. [8]

Use of some sort of keyed database system is almost essential for large MMDF systems, since the table lookup overhead is unbearable otherwise. Currently DBM is the only readily available alternative and it does seem to work well, but it is running out of steam, particularly due to its limited record length. A more flexible replacement for this package would be welcome.

The top of the MMDF directory tree contains the directories mentioned above, the **submit** and **deliver** programs, and a few maintenance programs. If the site is polled for PhoneNet mail then the "**slave**" program is normally also located here. The **slave** program is used as the remote site's login shell and acts much as **uucico** in managing the link level communications. It in turn calls upon **submit** and **deliver** to send and receive mail.

Submit

For all the changes to its internals, the external interface of **submit** has changed remarkably little since MMDFI. The only notable external changes to **submit** are that it now processes domain-style addresses (both forward and reverse!) and both RFC822 and RFC733 format addresses. [9] A couple of new options have been added to **submit** to give some control over the handling of returned mail in the event that a message cannot be delivered. This is useful if the user is a program and does not care if the message is undeliverable! It is also now possible to feed a bare message to **submit** and tell **submit** to find all the addresses and show them and their validity on a one-by-one basis. This makes it convenient to feed mail to **submit** from a smart mail composer that doesn't want to know how to parse addresses (a major task these days!).

The **submit** program can operate in one of two modes. In *protocol* mode, **submit** accepts options, a return address, and optionally a list of addresses for each message, followed by the message text. Multiple messages can be submitted one after another without reinvoking the **submit** process.

7 See the manual section on the MMDF database (queue.5) for more information.

8 The DBM package is a set of simple hashed database access routines that were distributed with V7 Unix and are still widely used.

9 D. Crocker, J. Vittal, K. Pogran, D. Henderson, "RFC733 - Standard for the Format of ARPA Network Text Message", Network Information Center, SRI International, November 1977.

D. Crocker, "RFC822 - Standard for the Format of ARPA Network Text Message", Network Information Center, SRI International, August 1982.

Each address is individually acknowledged. If there is an error, the submission of that letter is aborted and a new submission may be made. In *one-shot* mode a single message is submitted on the standard input. As in *protocol* mode, it can be preceded by options and addresses, or the options can be given on the command line and the addresses taken from the message text.

The internal address verification process of **submit** has changed greatly since MMDFI. Most of the changes have been made to properly support domain style addresses. Additional changes were made to support per-channel and per-user access controls. While **submit** is checking each address, regardless of origin, it is also compiling the address list for the message. Each address list entry contains the destination domain, [10] the destination mailbox, and the channel in whose channel table **submit** first found the destination host. The lookup is somewhat complicated. The destination domain is looked up in the domain tables; the first entry found is used. Each domain table entry has associated with it the routing to be used to reach that domain/host. Normally this is just the name of the host itself if the host is directly accessible, but it can be a sequence of hosts if the destination is not directly accessible. This "routing host" is then looked up in the channel tables to find a channel which can reach it. The routing host specifications and the entries in the channel tables are always full domain specifications, so as to be unambiguous.

Authorization is checked after a valid channel for a host is found. Access to send via a given channel depends on the originating channel and/or the submitting user. If access to a given channel is denied, **submit** will continue to look at subsequent channels to see if some other channel has access to the same host and is authorized. This mechanism is commonly used to restrict access to expensive transport systems or to restrict message transfer between channels representing private and public data networks. It can also be used to restrict relaying of messages between two "controlled-access" networks.

Deliver

The **deliver** process changed little until late 1984, when a dead-host caching facility and advanced retry mechanisms were added to **deliver**. Until then, the major changes to **deliver** were bug fixes and changes to enhance error recovery. Most notably, the mechanism to return mail was made much more persistent. In MMDFI, if a message could not be returned to the sender, it was "dropped on the floor". This is not acceptable. The new mechanism first tries to return to the sender; if that fails, an attempt is made to send the message to the local "Postmaster" address. If this too fails, then **deliver** tries to write the message into a "dead letter" file (usually /usr/mmdf/lock/home/DeadLetters). The last resort is to scream loudly into the log files.

The recent changes to **deliver** are designed to reduce **deliver**'s load on a system that handles a large amount of mail. The first change is to move the dead-host caching function out of the channels (currently only SMTP) and into **deliver**. This has two advantages. First, **deliver** no longer has to hand every address to the channel to find out that the host is dead. This was expensive, since communications between **deliver** and the channel are interactive using pipes and thus involved a great deal of context-switching and pipe I/O. Second, the dead-host caching is now a generally available facility that can be used by any channel without duplication of code.

The second change is to add a mechanism for storing the retry history for each dead host on each channel including retry count and last retry time. From this information it is possible to implement intelligent retry strategies using exponential backoff and maximal retry times. This greatly reduces the overhead caused by recalcitrant hosts that are unavailable over a long period of time. The retry history change is conditionally compiled into deliver since it can, by design, use quite a bit of memory if there are a lot of dead hosts or pending messages (or both!). Machines with a relatively limited address space may not be able to use this feature.

10 By "domain" we mean the full domain specification of a host, e.g. VAX1.UDEL.ARPA.

The Local Channel

The local channel remained unchanged until late 1983, when a major reworking of the channel was done by BRL. The old local channel could handle delivery in three basic ways. The first and most common was to deliver directly to the user's default mailbox, appending the message to the end. Second, the user could put a program in his private bin directory called "rcvmail" that would be called by the local channel to handle delivery of the message. If the user program (rcvmail) did not complete successfully, a standard delivery was made instead.

In early versions of the local channel for MMDFII, Dave Crocker added support for mailing to files and pipes, but the original version had a number of security problems, mostly due to **submit**, so the capability was not much used.

The latest version of the local channel has kept the alias-file-originated delivery to files and pipes, and the changes Steve Kille has made to the **submit** program have also made this facility reliable from a security standpoint. The rcvmail mechanism has been totally scrapped in favor of a more general and powerful mechanism which I will call "mail delivery files".

Mail delivery files were designed to give the user as much flexibility over how his mail was delivered as possible without opening security holes. The mail system was changed to allow local addresses to have a suffix appended which consists of an '=' and any simple text; this suffix is totally ignored except when using mail delivery files. Each user may create a ".maildelivery" file in his home directory which contains one or more delivery specifications. A delivery specification has five parts, separated by field separators, *<FS>*, which may be tabs, spaces or commas ",".

field <FS> pattern <FS> action <FS> A/R/

The *field* is the name of a field that is to be searched for a pattern. Currently supported fields are *From*, *To*, *Subject*, *Sender, and Cc*, plus three special fields, *addr*, *, and *. The *addr* field is used to match against the address being delivered to *including* the suffix (e.g. "dpk=unixwizards"). If the user subscribes to different lists with different suffixes he can use his mail delivery file to segregate his mail by source. To do this based on the message text alone is impossible to do right 100% of the time. The *default* field matches if the message has not been delivered by any of the preceding lines in the ".maildelivery" file. The * field always matches, regardless of any other action.

The *pattern* is some sequence of characters that may be matched in the *field*. *Case is not significant, and multiple fields of the same name are concatenated, separated by spaces. If the field does not need as pattern, a dash (-) or similar symbol is usually inserted to show that the field is present but not used.*

The *action* is "file" or ">", "pipe" or "|", or "destroy". "File" or ">" appends the message in standard mailbox format to the file specified in the optional string. "Pipe" or "|" causes the program in the optional string to be run with the message available on the standard input. "Destroy" causes the mail to be thrown away silently. This is useful if you go away on a long trip and don't want to unsubscribe to lists, but also don't want to come home to several thousand messages.

The *A/R/?* flag is a single character: 'A' for accept, 'R' for reject, or '?' for accept if not delivered yet. This flag indicates whether the action, if successful, is sufficient to mark the message as delivered. If the message is undelivered at the end of the .maildelivery file, the local channel next consults a system-wide file, such as */usr/lib/maildevliery*. If the message is still undelivered at the end of the system-wide file, a standard delivery is made to the default mailbox. This protects against mail being lost due to lack of foresight or errors in the maildelivery files.

The file is always read completely so that several matches can be made, and several actions taken. For example, the user could have a TTY alert message sent to his terminal and also have the message resent to his new home machine by the following .maildelivery file:

```
addr   dpk   pipe  R  "/usr/mmdf/mailutils/ttyalert"
addr   dpk   pipe  A  "/usr/brl/bin/resend dpk@brl-vgr.arpa"
```

The last line, if completed without error (a return code of 0 from resend), would mark the message as delivered because of the A (accept) flag in the fourthcolumn.

The List Channel

The list channel was developed as the result of BRL's experience in managing **large** Internet mailing lists. Two major problems were discovered with dealing with mailing lists in MMDFI. First, since **submit** always verifies all known addresses for a message at submission time, if you were on the machine with a large list and submitted mail to the list you would have to wait for every address on that list to be verified. On a busy machine with a list of hundreds of addresses, this could take five or ten minutes. Annoying as this may be for people, the situation was worse for mail submitted over communications channels like TCP/IP. The remote end would continually time out before the message had been completely verified so the message could never be sent.

The second problem with large lists was that there were always rejected mail notices going back to those least able to do anything about the mail problem, the original sender of the message. What was really desired was a method to try to have returned mail go to the list maintainer instead of the message's original sender.

The solution to both of the problems is embodied in the list channel. This is a channel with an incestuous relationship to **submit**, **deliver**, and the alias file. Use of the list channel is best described in parallel with the special entries for the list channel in the alias file. If we were maintaining a large list called "biglist", the following entries would be in the alias file:

biglist:	biglist-outbound@list-processor
biglist-outbound:	</usr/mmdf/lists/biglist-file
biglist-request:	maintainer

The pseudo-host "list-processor" has its own domain table and its own host table but represents no actual host. If someone submits mail to biglist, **submit** will find the alias entry and upon finding that it's not a local address, will queue it to the host "list-processor", so verification is complete after only one lookup. Unknown to **deliver**, the list channel simply calls **submit** and feeds it the aliased addresses, "biglist-outbound". This time the actual verification is done on the contents of the address list "biglist-file". Since the list channel is processed by a background daemon, no one is forced to wait through the verification process except the background daemon itself, which doesn't care how long it takes so long as it completes.

The list channel also performs another function to try to eliminate the problem of failed mail messages. For each address given to it, (normally just one), the list channel sees if there is a matching "*listname*-request" entry in the alias table. It knows enough to try stripping any "–outbound"'s from the name first though. If a "–request" entry is found, then that address is substituted instead of the original return address. The message text is *not* altered, but the new return address is recorded for use when resending the message. The new return address is supplied in SMTP "MAIL FROM:<*address*>" commands and any other situations where the return address is directly specifiable.

The changing of the return address is useful only if mail is rejected when submitted to the foreign host or if that host is smart enough to keep the return address information around. Many hosts do not maintain this information, and many of the same hosts are also problematic in that they will completely accept a message containing total garbage and decide to tell you about it later. This is precisely what MMDF tries to avoid by submission-time verification.

The BBOARDS Channel

Sites that run the MH message system, version mh.5, may install a *bboards* channel which delivers messages from interest-group mailing lists to a special "bboards" directory . The bboards software, which is compatible with the MH message system, keeps track of which messages have been seen by individual users, and allows designated bboards managers to control the size and access for different bboards. [11]

11 See the *man* entry mh-gen.8 in the MH distribution. It is important to note that the choice to install "bboards" must be made when MMDF is generated. The "news" facility mentioned in the MH documentation is not supported by CSNET.

The UUCP Channel

The task of integrating UUCP mail into MMDF was a prime goal for BRL. Our users would not tolerate having to use two radically different mail interfaces for two different kinds of mail connections. We decided to write a channel to interface to the UUCP mail world that would take care of the necessary format conversions to allow mail to traverse the two mail worlds. The channel has two parts. The input portion of the channel is the program **/bin/rmail** which is executed by the UUCP program **uuxqt** when mail is being delivered. The output portion is a standard channel that invokes the UUCP system after reformatting the message.

The **rmail** program has been totally rewritten to interface to MMDF. **Rmail**'s primary task is to collect and reformat the address strings in the message. To reformat each address, **rmail** uses the UUCP channel table to determine what hosts are known to this host and shortens an host!host!host! string down to the single most distant host we know about and any subsequent hosts we do not know. For example knownA!knownB!knownC!unknown1!unknown2!user would become knownC!unknown1!unknown2!user. It then converts this to an RFC822 style address by putting the unknown hosts and the user in the local part and putting the known host with a domain in the domain portion, e.g. unknown1!unknown2!user@knownC.UUCP. If all the hosts are known, then only the user is left in the local part, and the address winds up being user@known.UUCP. Since you always know the hosts you talk to, you can build any arbitrary UUCP path by simply saying arbitrary-host-path@neighbor.UUCP.

Rmail is prepared to accept destination addresses in two forms. If the addressee is just another UUCP host addressed using host!host!... notation, then **rmail** forwards the letter via UUCP without header munging since the destination host may not support RFC822 style mail. An addressee of the form user@domain will cause the message to be fed to **submit** and into MMDF proper where the message can be delivered to another UUCP site or any other site accessible via MMDF. **Rmail** will reformat the message header in the latter case to conform as much as possible with the RFC822 specifications.

The outbound portion of the UUCP channel is a MMDF channel program called "uucp" which is invoked by **deliver**. The job of this program is much easier since all it must do is reformat the "From:" line to be compatible with UUCP mail. The outbound channel must also reformat the destination addresses which become arguments to **uux**. The outbound channel uses the same channel table that **rmail** used but performs the reverse action on the address so, for example, root@mcnc.UUCP becomes unc!duke!mcnc!root and this is then further divided to form the uux command "uux unc!rmail duke!mcnc!root" (assuming the channel table maps mcnc.UUCP into duke!unc!mcnc).

The UUCP channel would have to be classed as the only "flakey" portion of MMDF since some of these address transformation really need an advanced AI system to make an intelligent transformation. In general, though, the channel does a very good job and has little trouble with "normal" UUCP addresses.

Using Domain Name Servers

The use of domain name servers will have some interesting effects on the address verification aspects of mail submission. In the current system, all the information necessary for verification of addresses is in a local data base. When we are using name servers, we can no longer be guaranteed that all the needed information will be locally cached. In addition, we are not guaranteed that we will be able to reach all the necessary name servers at submission time (although duplicate name servers will make this possibility small). The **submit** program will call the local domain resolver to verify each address, and there will be some time limit in which to complete this task. The resolver will be expected to first consult the local cache of domain data and, if the information is not found, contact as many servers as necessary to resolve the address.

The possible lack of information will force us to provide a contingent submission queue for those messages that cannot be verified at submission time. This does not imply that there will we be no verification. We will verify that we at least know the top level domain of each address and verify each sub-domain when possible. If some sub-domain of the full address is known to be bogus, the address can be flushed. Knowing that we have authoritative information that a domain does not exist is just as important as knowing that it does exist.

A new channel, much like the list channel, will be used to process the partially-accepted address for a message. This channel will continually try to verify the address until it is known to be good or bad. It will have the message returned to the sender, with an explanation, if one or more addresses is bad. Most systems will run with a fairly rich cache of host information. For those systems which cannot afford to keep this information around, the submission time verification might be a considerable delay which would be unacceptable for a user interface. On these systems it will possible to force all message to be accepted for background verification (via the "verification" channel).

Conclusion

MMDFII had some early problems and as a result may have gotten some initial bad press, but MMDFII has shown that it is a capable mail system which is both robust and able to handle very large mail loads. There now exist a growing number of tools to analyze and manage large flows of mail in a MMDF system. These tools include status programs, sophisticated logging, and log analysis programs. Because of the separation of mail into separate queues, multiprocessing of the mail queues is not only possible, but routinely used to both increase throughput and decrease delays. MMDF is also a flexible system. Runtime reconfiguration is simple, generally easy to understand, and can be done at any time. Since the MMDF core software is free of channel specific or network specific information, one can easily add additional channels for new networks or protocols without affecting the existing software. MMDFII represents a stable, production mail system, providing a strong base for the development of new network interconnections and mail handling environments which are essential in today's distributed computing environment.

The MMDFII software is available under license, free of charge (with the possible exception of a tape copy fee), for internal use only as follows: to U.S. Government agencies through the Ballistic Research Labs, to CSNET sites through the CSNET Coordination and Information Center at BBN, and to others through Prof. David Farber at the University of Delaware, Electrical Engineering and Computer Science Department. Commercial concerns interested in MMDF for other than internal use should contact Prof. Farber.

RN
READNEWS FRONT END

Larry Wall

NAME

rn - new read news program

SYNOPSIS

rn [options] [newsgroups]

DESCRIPTION

Rn is a replacement for the readnews(1) program that was written to be as efficient as possible, particularly in human interaction. *Rn* attempts to minimize the amount of "dead" time spent reading news—it tries to get things done while the user is reading or deciding whether to read, and attempts to get useful information onto the screen as soon as possible, highlighting spots that the eye makes frequent reference to, like subjects and previously read lines. Whether or not it's faster, it SEEMS faster.

If no newsgroups are specified, all the newsgroups which have unread news are displayed, and then the user is asked for each one whether he wants to read it, in the order in which the newsgroups occur in the *.newsrc* file. With a list of newsgroups, *rn* will start up in "add" mode, using the list as a set of patterns to add new newsgroups and restrict which newsgroups are displayed. See the discussion of the 'a' command on the newsgroup selection level.

Rn operates on three levels: the newsgroup selection level, the article selection level, and the paging level. Each level has its own set of commands, and its own help menu. At the paging level (the bottom level), *rn* behaves much like the *more*(1) program. At the article selection level, you may specify which article you want next, or read them in the default order, which is either in order of arrival on your system, or by subject threads. At the newsgroup selection level (the top level), you may specify which newsgroup you want next, or read them in the default order, which is the order that the newsgroups occur in your *.newsrc* file. (You will therefore want to rearrange your *.newsrc* file to put the most interesting newsgroups first. This can be done with the 'm' command on the Newsgroup Selection level. WARNING: invoking readnews/vnews (the old user interface) in any way (including as a news checker in your login sequence!) will cause your *.newsrc* to be disarranged again.)

On any level, at ANY prompt, an 'h' may be typed for a list of available commands. This is probably the most important command to remember, so don't you forget it. Typing space to any question means to do the normal thing. You will know what that is because every prompt has a list of several plausible commands enclosed in square brackets. The first command in the list is the one which will be done if you type a space. (All input is done in cbreak mode, so carriage returns should not be typed to terminate anything except certain multi-character commands. Those commands will be obvious in the discussion below because they take an argument.)

Upon startup, *rn* will do several things:

1. It will look for your *.newsrc* file, which is your list of subscribed-to newsgroups. If *rn* doesn't find a *.newsrc*, it will create one. If it does find one, it will back it up under the name ".oldnewsrc".
2. It will input your *.newsrc* file, listing out the first several newsgroups with unread news.
3. It will perform certain consistency checks on your *.newsrc*. If your *.newsrc* is out of date in any of several ways, *rn* will warn you and patch it up for you, but you may have to wait a little longer for it to start up.
4. *Rn* will next check to see if any new newsgroups have been created, and give you the opportunity to add them to your *.newsrc*.
5. *Rn* goes into the top prompt level—the newsgroup selection level.

Newsgroup Selection Level

In this section the words "next" and "previous" refer to the ordering of the newsgroups in your *.newsrc* file. On the newsgroup selection level, the prompt looks like this:

```
******** 17 unread articles in net.blurfl—read now? [ynq]
```

and the following commands may be given at this level:

y,SP	Do this newsgroup now.
.command	Do this newsgroup now, but execute *command* before displaying anything. The command will be interpreted as if given on the article selection level.
=	Do this newsgroup now, but list subjects before displaying articles.
n	Go to the next newsgroup with unread news.
N	Go to the next newsgroup.
p	Go to the previous newsgroup with unread news. If there is none, stay at the current newsgroup.
P	Go to the previous newsgroup.
-	Go to the previously displayed newsgroup (regardless of whether it is before or after the current one in the list).
1	Go to the first newsgroup.
^	Go to the first newsgroup with unread news.
$	Go to the end of the newsgroups list.
g newsgroup	Go to *newsgroup*. If it isn't currently subscribed to, you will be asked if you want to subscribe.
/pattern	Scan forward for a newsgroup matching *pattern*. Patterns do globbing like filenames, i.e., use ? to match a single character, * to match any sequence of characters, and [] to specify a list of characters to match. ("all" may be used as a synonym for "*".) Unlike normal filename globbing, newsgroup searching is not anchored to the front and back of the filename, i.e. "/jok" will find net.jokes. You may use ^ or $ to anchor the front or back of the search: "/^test$" will find newsgroup test and nothing else If you want to include newsgroups with 0 unread articles, append /r. If the newsgroup is not found between the current newsgroup and the last newsgroup, the search will wrap around to the beginning.
?pattern	Same as /, but search backwards.
u	Unsubscribe from current newsgroup.
l string	List newsgroups not subscribed to which contain the string specified.
L	Lists the current state of the *.newsrc*, along with status information.

Status	Meaning
<number>	Count of unread articles in newsgroup.
READ	No unread articles in newsgroup.
UNSUB	Unsubscribed newsgroup.
BOGUS	Bogus newsgroup.
JUNK	Ignored line in .newsrc (e.g. readnews "options" line).

(A bogus newsgroup is one that is not in the list of active newsgroups in the active file, which on most systems is /usr/lib/news/active.)

m name
: Move the named newsgroup somewhere else in the *.newsrc*. If no name is given, the current newsgroup is moved. There are a number of ways to specify where you want the newsgroup—type h for help when it asks where you want to put it.

c
: Catch up—mark all unread articles in this newsgroup as read.

o pattern
: Only display those newsgroups whose name matches *pattern*. Patterns are the same as for the '/' command. Multiple patterns may be separated by spaces, just as on the command line. The restriction will remain in effect either until there are no articles left in the restricted set of newsgroups, or another restriction command is given. Since *pattern* is optional, 'o' by itself will remove the restriction.

a pattern
: Add new newsgroups matching *pattern*. Newsgroups which are already in your *.newsrc* file, whether subscribed to or not, will not be listed. If any new newsgroups are found, you will be asked for each one whether you would like to add it. After any new newsgroups have been added, the 'a' command also restricts the current set of newsgroups just like the 'o' command does.

&
: Print out the current status of command line switches and any newsgroup restrictions.

&switch {switch}
: Set additional command line switches.

&&
: Print out the current macro definitions.

&&keys commands
: Define additional macros.

!command
: Escape to a subshell. One exclamation mark (!) leaves you in your own news directory. A double exclamation mark (!!) leaves you in the spool directory for news, which on most systems is /usr/spool/news. The environment variable SHELL will be used if defined. If *command* is null, an interactive shell is started.

q
: Quit.

x
: Quit, restoring .newsrc to its state at startup of *rn*. The .newsrc you would have had if you had exited with 'q' will be called .newnewsrc, in case you didn't really want to type 'x'.

^K
: Edit the global KILL file. This is a file which contains /pattern/j commands (one per line) to be applied to every newsgroup as it is started up, that is, when it is selected on the newsgroup selection level. The purpose of a KILL file is to mark articles as read on the basis of some set of patterns. This saves considerable wear and tear on your 'n' key. There is also a local KILL file for each newsgroup. Because of the overhead involved in searching for articles to kill, it is better if possible to use a local KILL file. Local KILL files are edited with a '^K' on the article selection level. There are also automatic ways of adding search commands to the local KILL file—see the 'K' command and the K search modifier on the article selection level.

: If either of the environment variables VISUAL or EDITOR is set, the specified editor will be invoked; otherwise a default editor (normally vi(1)) is invoked on the KILL file.

Article Selection Level

On the article selection level, *rn* selects (by default) unread articles in numerical order (the order in which articles have arrived at your site). If you do a subject search (^N), the default order is modified to be numerical order within each subject thread. You may switch back and forth between numerical order and subject thread order at will. The –S switch can be used to make subject search mode the default.

On the article selection level you are *not* asked whether you want to read an article before the article is displayed; rather, *rn* simply displays the first page (or portion of a page, at low baud rates) of the article and asks if you want to continue. The normal article selection prompt comes at the END of the article (though article selection commands can be given from within the middle of the article (the pager level) also). The prompt at the end of an article looks like this:

End of article 248 (of 257)—what next? [npq]

The following are the options at this point:

n,SP	Scan forward for next unread article. (Note: the 'n' (next) command when typed at the end of an article does not mark the article as read, since an article is automaticaly marked as read after the last line of it is printed. It is therefore possible to type a sequence such as 'mn' and leave the article marked as unread. The fact that an article is marked as read by typing 'n', 'N', '^N', 's', or 'S' within the MIDDLE of the article is in fact a special case.)
N	Go to the next article.
^N	Scan forward for the next article with the same subject, and make ^N default (subject search mode).
p	Scan backward for previous unread article. If there is none, stay at the current article.
P	Go to the previous article.
–	Go to the previously displayed article (regardless of whether that article is before or after this article in the normal sequence).
^P	Scan backward for the previous article with the same subject, and make ^N default (subject search mode).
^R	Restart the current article.
v	Restart the current article verbosely, displaying the entire header.
^L	Refresh the screen.
^X	Restart the current article, and decrypt as a rot13 message.
X	Refresh the screen, and decrypt as a rot13 message.
b	Back up one page.
q	Quit this newsgroup and go back to the newsgroup selection level.
^	Go to the first unread article.
$	Go to the last article (actually, one past the last article).
number	Go to the numbered article.

range{,range} command{:command}

Apply a set of commands to a set of articles. A range consists of either <article number> or <article number>–<article number>. A dot '.' represents the current article, and a dollar sign '$' represents the last article.

Applicable commands include 'm' (mark as unread), 'M' (delayed mark as unread), 'j' (mark as read), "s dest" (save to a destination), "!command" (shell escape), "=" (print the subject) and "C" (cancel).

j
: Junk the current article—mark it as read. If this command is used from within an article, you are left at the end of the article, unlike 'n', which looks for the next article.

m
: Mark the current article as still unread. (If you are in subject search mode you probably want to use M instead of m. Otherwise the current article may be selected as the beginning of the next subject thread.)

M
: Mark the current article as still unread, but not until the newsgroup is exited. Until then, the current article will be marked as read. This is useful for returning to an article in another session, or in another newsgroup.

/pattern
: Scan forward for article containing *pattern* in the subject. See the section on Regular Expressions. Together with the escape substitution facility described later, it becomes easy to search for various attributes of the current article, such as subject, article ID, author name, etc. The previous pattern can be recalled with "<esc>/". If *pattern* is omitted, the previous pattern is assumed.

/pattern/h
: Scan forward for article containing *pattern* in the header.

/pattern/a
: Scan forward for article containing *pattern* anywhere in article.

/pattern/r
: Scan read articles also.

/pattern/c
: Make search case sensitive. Ordinarily upper and lower case are considered the same.

/pattern/modifiers:command{:command}
: Apply the commands listed to articles matching the search command (possibly with h, a, or r modifiers). Applicable commands include 'm' (mark as unread), 'M' (delayed mark as unread), 'j' (mark as read), "s dest" (save to a destination), "!command" (shell escape), "=" (print the subject) and "C" (cancel). If the first command is 'm' or 'M', modifier r is assumed. A K may be included in the modifiers (not the commands) to cause the entire command (sans K) to be saved to the local KILL file, where it will be applied to every article that shows up in the newsgroup.

  For example, to save all articles in a given newsgroup to the line printer and mark them read, use "/^/|lpr:j". If you say "/^/K|lpr:j", this will happen every time you enter the newsgroup.

?pattern
: Scan backward for article containing *pattern* in the subject. May be modified as the forward search is: ?pattern?modifiers[:commands]. It is likely that you will want an r modifier when scanning backward.

k
: Mark as read all articles with the same subject as the current article. (Note: there is no single character command to temporarily mark as read (M command) articles matching the current subject. That can be done with "/<esc>s/M", however.)

K
: Do the same as the k command, but also add a line to the local KILL file for this newsgroup to kill this subject every time the newsgroup is started up. For a discussion of KILL files, see the '^K' command below. See also the K modifier on searches above.

^K Edit the local KILL file for this newsgroup. Each line of the KILL file should be a command of the form /pattern/j. (With the exception that *rn* will insert a line at the beginning of the form "THRU <number>", which tells *rn* the maximum article number that the KILL file has been applied to. You may delete the THRU line to force a rescan of current articles.) You may also have reason to use the m, h, or a modifiers. Be careful with the M modifier in a kill file—there are more efficient ways to never read an article. You might have reason to use it if a particular series of articles is posted to multiple newsgroups. In this case, M would force you to view the article in a different newsgroup.

To see only newgroup articles in the control newsgroup, for instance, you might put

```
/^/j
/newgroup/m
```

which kills all subjects not containing "newgroup". You can add lines automatically via the K command and K search modifiers, but editing is the only way to remove lines. If either of the environment variables VISUAL or EDITOR is set, the specified editor will be invoked; otherwise a default editor (normally vi) is invoked on the KILL file.

The KILL file may also contain switch setting lines beginning with '&'. Additionally, any line beginning with 'X' is executed on exit from the newsgroup rather than on entrance. This can be used to set switches back to a default value.

r Reply through net mail. The environment variables MAILPOSTER and MAILHEADER may be used to modify the mailing behavior of *rn* (see environment section). If on a nonexistent article such as the "End of newsgroup" pseudo-article (which you can get to with a '$' command), invokes the mailer to nobody in particular.

R Reply, including the current article in the header file generated. (See 'F' command below). The YOUSAID environment variable controls the format of the attribution line.

f Submit a followup article. If on a nonexistent article such as the "End of newsgroup" pseudo-article (which you can get to with a '$' command), posts an original article (basenote).

F Submit a followup article, and include the old article, with lines prefixed either by ">" or by the argument to a -F switch. *Rn* will attempt to provide an attribution line in front of the quoted article, generated from the From: line of the article. Unfortunately, the From: line doesn't always contain the right name; you should double check it against the signature and change it if necessary, or you may have to apologize for quoting the wrong person. The environment variables NEWSPOSTER, NEWSHEADER and ATTRIBUTION may be used to modify the posting behavior of *rn* (see environment section).

C Cancel the current article, but only if you are the contributor or superuser.

c Catch up in this newsgroup; i.e., mark all articles as read.

u Unsubscribe to this newsgroup.

s destination
Save to a filename or pipe using sh. If the first character of the destination is a vertical bar, the rest of the command is considered a shell command to which the article is passed through standard input. The command is subject to filename expansion. (See also the environment variable PIPESAVER.) If the destination does not begin with a vertical bar, the rest of the command is assumed to be a filename of some

sort. An initial tilde '~' will be translated to the name of the home directory, and an initial environment variable substitution is also allowed. If only a directory name is specified, the environment variable SAVENAME is used to generate the actual name. If only a filename is specified (i.e. no directory), the environment variable SAVEDIR will be used to generate the actual directory. If nothing is specified, then obviously both variables will be used. Since the current directory for rn while doing a save command is your private news directory, saying "s ./filename" will force the file to your news directory. Save commands are also run through % interpretation, so that you can say "s %O/filename" to save to the directory you were in when you ran *rn*, and "s %t" to save to a filename consisting of the Internet address of the sender.

After generating the full pathname of the file to save to, *rn* determines if the file exists already, and if so, appends to it. *Rn* will attempt to determine if an existing file is a mailbox or a normal file, and save the article in the same format. If the output file does not yet exist, *rn* will by default ask you which format you want, or you can make it skip the question with either the -M or -N switch. If the article is to be saved in mailbox format, the command to do so is generated from the environment variable MBOXSAVER. Otherwise, NORMSAVER is used.

S destination
: Save to a filename or pipe using a preferred shell, such as csh. Which shell is used depends first on what you have the environment variable SHELL set to, and in the absence of that, on what your news administrator set for the preferred shell when he or she installed *rn*.

| command
: Shorthand for "s | command".

w destination
: The same as "s destination", but saves without the header.

W destination
: The same as "S destination", but saves without the header.

& Print out the current status of command line switches.

&switch {switch}
: Set additional command line switches.

&& Print out current macro definitions.

&&keys commands
: Define an additional macro.

!command
: Escape to a subshell. One exclamation mark (!) leaves you in your own news directory. A double exclamation mark (!!) leaves you in the spool directory of the current newsgroup. The environment variable SHELL will be used if defined. If *command* is null, an interactive shell is started.

You can use escape key substitutions described later to get to many run-time values. The command is also run through % interpretation, in case it is being called from a range or search command.

= List subjects of unread articles.

Print last article number.

Pager Level

At the pager level (within an article), the prompt looks like this:

—MORE—(17%)

and a number of commands may be given:

SP	Display next page.
x	Display next page and decrypt as a rot13 message.
d,^D	Display half a page more.
CR	Display one more line.
q	Go to the end of the current article (don't mark it either read or unread). Leaves you at the "What next?" prompt.
j	Junk the current article. Mark it read and go to the end of the article.
^L	Refresh the screen.
X	Refresh the screen and decrypt as a rot13 message.
b,^B	Back up one page.
gpattern	Goto (search forward for) *pattern* within current article. Note that there is no space between the command and the pattern. If the pattern is found, the page containing the pattern will be displayed. Where on the page the line matching the pattern goes depends on the value of the **–g** switch. By default the matched line goes at the top of the screen.
G	Search for g pattern again.
^G	This is a special version of the 'g' command that is for skipping articles in a digest. It is equivalent to setting "–g4" and then executing the command "g^Subject:".
TAB	This is another special version of the 'g' command that is for skipping inclusions of older articles. It is equivalent to setting "–g4" and then executing the command "g^[^c]", where *c* is the first character of the last line on the screen. It searches for the first line that doesn't begin with the same character as the last line on the screen.
!command	Escape to a subshell.

The following commands skip the rest of the current article, then behave just as if typed to the "What next?" prompt at the end of the article. See the documentation at the article selection level for these commands.

```
# $ & / = ? c C f F k K ^K m M r R ^R u v Y ^
number
range{,range} command{:command}
```

The following commands also skip to the end of the article, but have the additional effect of marking the current article as read:

```
n N ^N s S | w W
```

Miscellaneous facts about commands

An 'n' typed at either the "Last newsgroup" prompt or a "Last article" prompt will cycle back to the top of the newsgroup or article list, whereas a 'q' will quit the level. (Note that 'n' does not mean "no", but rather "next".) A space will of course do whatever is shown as the default, which will vary depending on whether rn thinks you have more articles or newsgroups to read.

The 'b' (backup page) command may be repeated until the beginning of the article is reached. If *rn* is suspended (via a ^Z), then when the job is resumed, a refresh (^L) will automatically be done (Berkeley-type systems only). If you type a command such as '!' or 's' which takes you from the middle of the article to the end, you can always get back into the middle by typing '^L'.

In multi-character commands such as '!', 's', '/', etc, you can interpolate various run-time values by typing escape and a character. To find out what you can interpolate, type escape and 'h', or check out the single character % substitutions for environment variables in the Interpretation and Interpolation section, which are the same. Additionally, typing a double escape will cause any % substitutions in the string already typed in to be expanded.

Options

Rn has a nice set of options to allow you to tailor the interaction to your liking. (You might like to know that the author swears by "-e -m -S -/".) These options may be set on the command line, via the RNINIT environment variable, via a file pointed to by the RNINIT variable, or from within rn via the & command. Options may generally be unset by saying "+switch". Options include:

-c checks for news without reading news. If a list of newsgroups is given on the command line, only those newsgroups will be checked; otherwise all subscribed-to newsgroups are checked. Whenever the **-c** switch is specified, a non-zero exit status from *rn* means that there is unread news in one of the checked newsgroups. The **-c** switch does not disable the printing of newsgroups with unread news; this is controlled by the **-s** switch. (The **-c** switch is not meaningful when given via the & command.)

-C<number>
tells *rn* how often to checkpoint the *.newsrc*, in articles read. Actually, this number says when to start thinking about doing a checkpoint if the situation is right. If a reasonable checkpointing situation doesn't arise within 10 more articles, the *.newsrc* is checkpointed willy-nilly.

-d<directory name>
sets the default save directory to something other than ~/News. The directory name will be globbed (via csh) if necessary (and if possible). Articles saved by *rn* may be placed in the save directory or in a subdirectory thereof depending on the command that you give and the state of the environment variables SAVEDIR and SAVENAME. Any KILL files (see the K command in the Article Selection section) also reside in this directory and its subdirectories, by default. In addition, shell escapes leave you in this directory.

-D<flags>
enables debugging output. See common.h for flag values. Warning: normally *rn* attempts to restore your *.newsrc* when an unexpected signal or internal error occurs. This is disabled when any debugging flags are set.

-e causes each page within an article to be started at the top of the screen, not just the first page. (It is similar to the **-c** switch of *more*(1).) You never have to read scrolling text with this switch. This is helpful especially at certain baud rates because you can start reading the top of the next page without waiting for the whole page to be printed. It works nicely in conjuction with the **-m** switch, especially if you use half-intensity for your highlight mode. See also the -L switch.

-E<name>=<val>
sets the environment variable <name> to the value specified. Within *rn*, "&-ESAVENAME=%t" is similar to "setenv SAVENAME '%t'" in *csh*, or "SAVENAME='%t'; export SAVENAME" in *sh*. Any environment variables set with

-E will be inherited by subprocesses of *rn*.

-F<string>
sets the prefix string for the 'F' followup command to use in prefixing each line of the quoted article. For example, "-F<tab>" inserts a tab on the front of each line (which will cause long lines to wrap around, unfortunately), "-F>>>>" inserts ">>>>" on every line, and "-F" by itself causes nothing to be inserted, in case you want to reformat the text, for instance. The initial default prefix is ">".

-g<line>
tells *rn* which line of the screen you want searched-for strings to show up on when you search with the 'g' command within an article. The lines are numbered starting with 1. The initial default is "-g1", meaning the first line of the screen. Setting the line to less than 1 or more than the number of lines on the screen will set it to the last line of the screen.

-h<string>
hides (disables the printing of) all header lines beginning with *string*. For instance, -hexp will disable the printing of the "Expires:" line. Case is insignificant. If <string> is null, all header lines except Subject are hidden, and you may then use **+h** to select those lines you want to see. You may wish to use the baud-rate switch modifier below to hide more lines at lower baud rates.

-H<string>
works just like **-h** except that instead of setting the hiding flag for a header line, it sets the magic flag for that header line. Certain header lines have magic behavior that can be controlled this way. At present, the following actions are caused by the flag for the particular line: the Newsgroups line will only print when there are multiple newsgroups, the Subject line will be underlined, and the Expires line will always be suppressed if there is nothing on it. In fact, all of these actions are the default, and you must use +H to undo them.

-i=<number>
specifies how long (in lines) to consider the initial page of an article—normally this is determined automatically depending on baud rate. (Note that an entire article header will always be printed regardless of the specified initial page length. If you are working at low baud rate and wish to reduce the size of the headers, you may hide certain header lines with the **-h** switch.)

-l disables the clearing of the screen at the beginning of each article, in case you have a bizarre terminal.

-L tells *rn* to leave information on the screen as long as possible by not blanking the screen between pages, and by using clear to end-of-line. (The *more*(1) program does this.) This feature works only if you have the requisite termcap capabilities. The switch has no effect unless the **-e** switch is set.

-m=<mode>
enables the marking of the last line of the previous page printed, to help the user see where to continue reading. This is most helpful when less than a full page is going to be displayed. It may also be used in conjunction with the **-e** switch, in which case the page is erased, and the first line (which is the last line of the previous page) is highlighted. If **-m=s** is specified, the standout mode will be used, but if **-m=u** is specified, underlining will be used. If neither =s or =**u** is specified, standout is the default. Use +**m** to disable highlighting.

-M forces mailbox format in creating new save files. Ordinarily you are asked which format you want.

-N forces normal (non-mailbox) format in creating new save files. Ordinarily you are asked which format you want.

-r causes *rn* to restart in the last newsgroup read during a previous session with *rn*. It is equivalent to starting up normally and then getting to the newsgroup with a g command.

-s with no argument suppresses the initial listing of newsgroups with unread news, whether **-c** is specified or not. Thus **-c** and **-s** can be used together to test "silently" the status of news from within your *.login* file. If **-s** is followed by a number, the initial listing is suppressed after that many lines have been listed. Presuming that you have your *.newsrc* sorted into order of interest, **-s5** will tell you the 5 most interesting newsgroups that have unread news. This is also a nice feature to use in your *.login* file, since it not only tells you whether there is unread news, but also how important the unread news is, without having to wade through the entire list of unread newsgroups. If no **-s** switch is given **-s5** is assumed, so just putting "rn -c" into your .login file is fine.

-S<number>
causes *rn* to enter subject search mode (^N) automatically whenever a newsgroup is started up with <number> unread articles or more. Additionally, it causes any 'n' typed while in subject search mode to be interpreted as '^N' instead. (To get back out of subject search mode, the best command is probably '^'.) If <number> is omitted, 3 is assumed.

-t puts *rn* into terse mode. This is more cryptic but useful for low baud rates. (Note that your system administrator may have compiled *rn* with either verbose or terse messages only to save memory.) You may wish to use the baud-rate switch modifier below to enable terse mode only at lower baud rates.

-T allows you to type ahead of rn. Ordinarily rn will eat typeahead to prevent your autorepeating space bar from doing a very frustrating thing when you accidentally hold it down. If you don't have a repeating space bar, or you are working at low baud rate, you can set this switch to prevent this behavior. You may wish to use the baud-rate switch modifier below to disable typeahead only at lower baud rates.

-v sets verification mode for commands. When set, the command being executed is displayed to give some feedback that the key has actually been typed. Useful when the system is heavily loaded and you give a command that takes a while to start up.

-/ sets SAVEDIR to "%p/%c" and SAVENAME to "%a", which means that by default articles are saved in a subdirectory of your private news directory corresponding to the name of the the current newsgroup, with the filename being the article number. +/ sets SAVEDIR to "%p" and SAVENAME to "%^C", which by default saves articles directly to your private news directory, with the filename being the name of the current newsgroup, first letter capitalized. (Either +/ or -/ may be default on your system, depending on the feelings of your news administrator when he, she or it installed *rn*.) You may, of course, explicitly set SAVEDIR and SAVENAME to other values—see discussion in the environment section.

Any switch may be selectively applied according to the current baud-rate. Simply prefix the switch with +speed to apply the switch at that speed or greater, and -speed to apply the switch at that speed or less. Examples: -1200-hposted suppresses the Posted line at 1200 baud or less; +9600-m enables marking at 9600 baud or more. You can apply the modifier recursively to itself also: +300-1200-t sets terse mode from 300 to 1200 baud.

Similarly, switches may be selected based on terminal type:

```
-=vt100+T              set +T on vt100
-=tvi920-ETERM=mytvi      get a special termcap entry
-=tvi920-ERNMACRO=%./.rnmac.tvi
```

	set up special keymappings
+=paper–v	set verify mode if not hardcopy

Some switch arguments, such as environment variable values, may require spaces in them. Such spaces should be quoted via ", ', or \ in the conventional fashion, even when passed via RNINIT or the & command.

Regular Expressions

The patterns used in article searching are regular expressions such as those used by *ed*(1). In addition, \w matches an alphanumeric character and \W a nonalphanumeric. Word boundaries may be matched by \b, and non-boundaries by \B. The bracketing construct \(... \) may also be used, and \digit matches the digit'th substring, where digit can range from 1 to 9. \0 matches whatever the last bracket match matched. Up to 10 alternatives may given in a pattern, separated by \|, with the caveat that \(... \| ... \) is illegal.

Interpretation and Interpolation

Many of the strings that *rn* handles are subject to interpretations of several types. Under filename expansion, an initial "~/" is translated to the name of your home directory, and "~name" is translated to the login directory for the user specified. Filename expansion will also expand an initial environment variable, and also does the backslash, uparrow and percent expansion mentioned below.

All interpreted strings go through backslash, uparrow and percent interpretation. The backslash escapes are the normal ones (such as \n, \t, \nnn, etc.). The uparrow escapes indicate control codes in the normal fashion. Backslashes or uparrows to be passed through should be escaped with backslash. The special percent escapes are similar to printf percent escapes. These cause the substitution of various run-time values into the string. The following are currently recognized:

%a	Current article number.
%A	Full name of current article (%P/%c/%a). (On a Eunice system with the LINKART option, %P/%c/%a returns the name of the article in the current newsgroup, while %A returns the real name of the article, which may be different if the current article was posted to multiple newsgroups.)
%b	Destination of last save command, often a mailbox.
%B	The byte offset to the beginning of the part of the article to be saved, set by the save command. The 's' and 'S' commands set it to 0, and the 'w' and 'W' commands set it to the byte offset of the body of the article.
%c	Current newsgroup, directory form.
%C	Current newsgroup, dot form.
%d	Full name of newsgroup directory (%P/%c).
%D	"Distribution:" line from the current article.
%f	"From:" line from the current article, or the "Reply-To:" line if there is one. This differs from %t in that comments (such as the full name) are not stripped out with %f.
%F	"Newsgroups:" line for a new article, constructed from "Newsgroups:" and "Followup-To:" lines of current article.
%h	Name of the header file to pass to the mail or news poster, containing all the information that the poster program needs in the form of a message header. It may also contain a copy of the current article. The format of the header file is controlled by the MAILHEADER and NEWSHEADER environment variables.

%H	Host name (your machine's name).
%i	"Message-I.D.:" line from the current article, with <> guaranteed.
%I	The reference indication mark (see the -F switch.)
%l	The news administrator's login name, if any.
%L	Login name (yours).
%m	The current mode of rn, for use in conditional macros.

i	Initializing.
n	Newsgroup selection level.
a	Article selection level (What next?).
p	Pager level (MORE prompt).
A	Add this newsgroup?
C	Catchup confirmation.
D	Delete bogus newsgroups?
M	Use mailbox format?
R	Resubscribe to this newsgroup?

Note that yes/no questions are all upper-case modes. If, for example, you wanted to disallow defaults on all yes/no questions, you could define the following macro:

```
\040   %(%m=[A-Z]?h: )
```

%M	The number of articles marked to return via the 'M' command. If the same article is Marked multiple times, "%M" counts it multiple times in the current implementation.
%n	"Newsgroups:" line from the current article.
%N	Full name (yours).
%o	Organization (yours).
%O	Original working directory (where you ran rn from).
%p	Your private news directory, normally ~/News.
%P	Public news spool directory, normally /usr/spool/news.
%r	Last reference on references line of current article (parent article id).
%R	References list for a new article, constructed from the references and article ID of the current article.
%s	Subject, with all Re's and (nf)'s stripped off.
%S	Subject, with one "Re:" stripped off.
%t	"To:" line derived from the "From:" and "Reply-To:" lines of the current article. This always returns an Internet format address.
%T	"To:" line derived from the "Path:" line of the current article to produce a uucp path.
%u	The number of unread articles in the current newsgroup.
%U	The number of unread articles in the current newsgroup, not counting the current article.
%x	The news library directory.
%X	The rn library directory.

%z The length of the current article in bytes.

%~ Your home directory.

%. The directory containing your dot files, which is your home directory unless the environment variable DOTDIR is defined when rn is invoked.

%$ Current process number.

%/ Last search string.

%% A percent sign.

%{name} or %{name–default}
The environment variable "name".

%[name]
The value of header line "Name:" from the current article. The "Name: " is not included. For example "%D" and "%[distribution]" are equivalent. The name must be spelled out in full.

%'command'
Inserts the output of the command, with any embedded newlines translated to space.

%"prompt"
Prints prompt on the terminal, then inputs one string, and inserts it.

%(test_text=pattern?then_text:else_text)
If *test_text* matches *pattern*, has the value *then_text*, otherwise *else_text*. The ":else_text" is optional, and if absent, interpolates the null string. The = may be replaced with != to negate the test. To quote any of the metacharacters ('=', '?', ':', or ')'), precede with a backslash.

%digit The digits 1 through 9 interpolate the string matched by the nth bracket in the last pattern match that had brackets. If the last pattern had alternatives, you may not know the number of the bracket you want—%0 will give you the last bracket matched.

Modifiers: to capitalize the first letter, insert '^': "%^C" produces something like "Net.jokes". Inserting '_' causes the first letter following the last '/' to be capitalized: "%_c" produces "net/Jokes".

ENVIRONMENT

The following environment variables are paid attention to by *rn*. In general the default values assumed for these variables by *rn* are reasonable, so if you are using *rn* for the first time, you can safely ignore this section. Note that the defaults below may not correspond precisely to the defaults on your system. To find the actual defaults you would need to look in config.h and common.h in the rn source directory, and the file INIT in the rn library.

Those variables marked (%) are subject to % interpolation, and those marked (~) are subject to both % interpolation and ~ interpretation.

ATTRIBUTION (%)
Gives the format of the attribution line in front of the quoted article included by an F command.

Default: In article %i %f writes:

CANCEL (~)
The shell command used to cancel an article.

Default: inews -h < %h

CANCELHEADER (%)
: The format of the file to pass to the CANCEL command in order to cancel an article.

  Default:
  Newsgroups: %n
  Subject: cmsg cancel %i
  References: %R
  Reply-To: %L@%H.UUCP (%N)
  Distribution: %D
  Organization: %o

  %i cancelled from rn.

DOTDIR
: Where to find your dot files, if they aren't in your home directory. Can be interpolated using "%.".

  Default: $HOME

EDITOR (~)
: The name of your editor, if VISUAL is undefined.

  Default: whatever your news administrator compiled in, usually vi.

FIRSTLINE (%)
: Controls the format of the line displayed at the top of an article. Warning: this may go away.

  Default: Article %a %(%U%M!=^00$?(%U more%(%M!=^0$? + %M Marked to return)\))in %C:, more or less.

HIDELINE
: If defined, contains a regular expression which matches article lines to be hidden, in order, for instance, to suppress quoted material. A recommended string for this purpose is "^>...", which *doesn't* hide lines with only '>', to give some indication that quoted material is being skipped. If you want to hide more than one pattern, you can use " | " to separate the alternatives. You can view the hidden lines by restarting the article with the 'v' command.

  There is some overhead involved in matching each line of the article against a regular expression. You might wish to use a baud-rate modifier to enable this feature only at low baud rates.

  Default: undefined

HOME
: Your home directory. Affects ~ interpretation, and the location of your dot files if DOTDIR is not defined.

  Default: $LOGDIR

KILLGLOBAL (~)
: Where to find the KILL file to apply to every newsgroup. See the '^K' command at the newsgroup selection level.

  Default: %p/KILL

KILLLOCAL (~)
: Where to find the KILL file for the current newsgroup. See the commands 'K' and '^K' at the article selection level, and the search modifier 'K'.

  Default: %p/%c/KILL

LOGDIR
: Your home directory if HOME is undefined. Affects ˜ interpretation, and the location of your dot files if DOTDIR is not defined.

  Default: none.

  Explanation: you must have either $HOME or $LOGDIR.

LOGNAME
: Your login name, if USER is undefined. May be interpolated using "%L".

  Default: value of getlogin().

MAILCALL (˜)
: What to say when there is new mail.

  Default: (Mail)

MAILFILE (˜)
: Where to check for mail.

  Default: /usr/spool/mail/%L

MAILHEADER (%)
: The format of the header file for replies. See also MAILPOSTER.

  Default:

```
To: %T
Subject: %(%i=^$?:Re: %S
Newsgroups: %n
In-Reply-To: %i)
%(%[references]!=^$?References\: %[references]
)Organization: %o
Cc:
Bcc: \n\n
```

MAILPOSTER (˜)
: The shell command to be used by the reply commands (r and R) in order to allow you to enter and deliver the response. *Rn* will not itself call upon an editor for replies—this is a function of the program called by *rn*. See also MAILHEADER.

  Default: Rnmail -h %h

MBOXSAVER (˜)
: The shell command to save an article in mailbox format.

  Default: %X/mbox.saver %A %P %c %a %B %C "%b" \
  "From: %T %'date'"

  Explanation: the first seven arguments are the same as for NORMSAVER. The eighth argument to the shell script is the new From: line for the article, including the posting date, derived either directly from the Posted: line, or not-so-directly from the Date: line. Header munging at its finest.

NAME
: Your full name. May be interpolated using "%N".

  Default: name from /etc/passwd, or ˜/.fullname.

NEWSHEADER (%)
: The format of the header file for followups. See also NEWSPOSTER.

  Default:

```
Newsgroups: %(%F=^$?%C:%F)
```

```
Subject: %(%S=^$?%"0ubject: ":Re: %S)
Summary:
Expires:
%(%R=^$?:References: %R
)Sender:
Reply-To: %L@%H.UUCP (%N)
Followup-To:
Distribution: %(%i=^$?%"0istribution: ":%D)
Organization: %o
Keywords: \n\n
```

NEWSPOSTER (~)

The shell command to be used by the followup commands (f and F) in order to allow you to enter and post a followup news article. *Rn* will not itself call upon an editor for followups—this is a function of the program called by *rn*. See also NEWSHEADER.

Default: Pnews -h %h

NORMSAVER (~)

The shell command to save an article in the normal (non-mailbox) format.

Default: %X/norm.saver %A %P %c %a %B %C "%b"

ORGANIZATION

Either the name of your organization, or the name of a file containing the name of your organization. May be interpolated using "%o".

Default: whatever your news administrator compiled in.

PAGESTOP

If defined, contains a regular expression which matches article lines to be treated as form-feeds. There are at least two things you might want to do with this. To cause page breaks between articles in a digest, you might define it as "^-------". To force a page break before a signature, you could define it as "^-- $". (Then, when you see "--" at the bottom of the page, you can skip the signature if you so desire by typing 'n' instead of space.) To do both, you could use "^--". If you want to break on more than one pattern, you can use " | " to separate the alternatives.

There is some overhead involved in matching each line of the article against a regular expression. You might wish to use a baud-rate modifier to enable this feature only at low baud rates.

Default: undefined

PIPESAVER (%)

The shell command to execute in order to accomplish a save to a pipe ("s | command" or "w | command"). The command typed by the user is substituted in as %b.

Default: %(%B=^0$?<%A:tail +%Bc %A |) %b

Explanation: if %B is 0, the command is "<%A %b", otherwise the command is "tail +%Bc %A | %b".

RNINIT

Default values for switches may be passed to *rn* by placing them in RNINIT. Any switch that is set in RNINIT may be overruled on the command line, or via the '&' command from within *rn*. Binary-valued switches that are set with "-switch" may be unset using "+switch".

If RNINIT begins with a '/' it is assumed to be the name of a file containing switches. If you want to set many environment variables but don't want to keep them all in your environment, or if the use of any of these variables conflicts with other programs, you can use this feature along with the -E switch to set the environment variables upon startup.

Default: " ".

RNMACRO (~)
The name of the file containing macros and key mappings. See the MACROS section.

Default: %./.rnmac

SAVEDIR (~)
The name of the directory to save to, if the save command does not specify a directory name.

Default:
If -/ is set: %p/%c
If +/ is set: %p

SAVENAME (%)
The name of the file to save to, if the save command contains only a directory name.

Default:
If -/ is set: %a
If +/ is set: %^C

SHELL The name of your preferred shell. It will be used by the '!', 'S' and 'W' commands.

Default: whatever your news administrator compiled in.

SUBJLINE (%)
Controls the format of the lines displayed by the '=' command at the article selection level.

Default: %s

TERM Determines which termcap entry to use, unless TERMCAP contains the entry.

TERMCAP
Holds either the name of your termcap file, or a termcap entry.

Default: /etc/termcap, normally.

USER Your login name. May be interpolated using "%L".

Default: $LOGNAME

VISUAL (~)
The name of your editor.

Default: $EDITOR

YOUSAID (%)
Gives the format of the attribution line in front of the quoted article included by an R command.

Default: In article %i you write:

MACROS

When *rn* starts up, it looks for a file containing macro definitions (see environment variable RNMACRO). Any sequence of commands may be bound to any sequence of keys, so you could remap your entire keyboard if you desire. Blank lines or lines beginning with # in the macro file are considered comments; otherwise *rn* looks for two fields separated by white space. The first field gives the sequence of keystrokes that trigger the macro, and the second field gives the sequence of commands to execute. Both fields are subject to % interpolation, which will also translate backslash and uparrow sequences. (The keystroke field is interpreted at startup time, but the command field is interpreted at macro execution time so that you may refer to % values in a macro.) For example, if you want to reverse the roles of carriage return and space in rn

```
^J      \040
^M      \040
\040    ^J
```

will do just that. By default, all characters in the command field are interpreted as the canonical *rn* characters, i.e. no macro expansion is done. Otherwise the above pair of macros would cause an infinite loop. To force macro expansion in the command field, enclose the macro call with ^(... ^) thusly:

```
@s      |mysavescript
@w      w^(@s^)
```

You can use the %() conditional construct to construct macros that work differently under different circumstances. In particular, the current mode (%m) of *rn* could be used to make a command that only works at a particular level. For example,

```
^[[O    %(%m=p?\040)
```

will only allow the macro to work at the pager level.

```
%(%{TERM}=vt100?^[[O)       /^J
```

will do the binding only if the terminal type is vt100, though if you have many of these it would be better to have separate files for each terminal.

If you want to bind a macro to a function key that puts a common garbage character after the sequence (such as the carriage return on the end of Televideo 920 function sequences), DO NOT put the carriage return into all the sequences or you will waste a CONSIDERABLE amount of internal storage. Instead of "^AF^M", put "^AF+1", which indicates to *rn* that it should gobble up one character after the F.

AUTHOR

Larry Wall <lwall@sdcrdcf.UUCP>
Regular expression routines are borrowed from emacs, by James Gosling.

FILES

%./.newsrc	status of your news reading
%./.oldnewsrc	backup copy of your *.newsrc* from start of session
%./.rnlock	lock file so you don't screw up your *.newsrc*
%./.rnlast	info from last run of rn
%./.rnsoft	soft pointers into /usr/lib/active to speed startup, synchronous with *.newsrc*
%./.rnhead	temporary header file to pass to a mailer or news poster
%./.rnmac	macro and keymap definitions
%p	your news save directory, usually ~/News

%x/active	the list of active newsgroups, usually /usr/lib/news/active
%P	the public news spool directory, usually /usr/spool/news
%X/INIT	system-wide default switches

SEE ALSO

newsrc(5), more(1), readnews(1), Pnews(1), Rnmail(1)

DIAGNOSTICS

Generally self-documenting, as they say.

BUGS

The **-h** switch can only hide header lines that *rn* knows about.

The '-' command doesn't cross newsgroup boundaries, and only undoes the last article selection.

If you edit your *.newsrc* while *rn* is running, *rn* will happily wipe out your changes when it decides to write out the *.newsrc* file.

Rn doesn't do certain things (like ordering articles on posting date) that the author feels should be handled by inews.

Marking of duplicate articles as read in cross-referenced newsgroups will not work unless the Xref patch is installed in inews.

If you get carried away with % or escape substitutions, you can overflow buffers.

There should be no fixed limit on the number of newsgroups.

Some of the more esoteric features may be missing on machines with limited address space.

NAME

Pnews - a program for posting news articles

SYNOPSIS

Pnews newsgroup title
or
Pnews -h headerfile [oldarticle]
or
Pnews

DESCRIPTION

Pnews is a friendly interface for posting news articles. It will ask several questions, then allow you to enter your article, and then post it using the inews(1) program. If you type h and a carriage return at any point, *Pnews* will tell you what it wants to know.

The -h form is used when invoked from *rn*. If your editor can edit multiple files, and you want the article to which you are replying to show up as an alternate file, define the environment variable NEWSPOSTER as "Pnews -h %h %A". You can also modify the the NEWSHEADER environment variable to change the header file that *rn* passes to Pnews.

ENVIRONMENT

AUTHORCOPY
: If defined, contains the name of a file to which the finished article will be appended.

  Default: article not saved

DOTDIR
: Where to find your dot files, if they aren't in your home directory. This is primarily for accounts which are shared by more than one person.

  Default: $HOME

EDITOR
: The editor you want to use, if VISUAL is undefined.

  Default: whatever your news administrator installed, usually vi.

HOME
: Your home directory.

  Default: $LOGDIR

LOGDIR
: Your home directory if HOME is undefined.

LOGNAME
: Your login name, if USER is undefined.

  Default: value of "whoami".

NAME
: Your full name.

  Default: name from /etc/passwd, or ~/.fullname.

ORGANIZATION
: Either the name of your organization, or the name of a file containing the name of your organization.

  Default: whatever your news administrator chose.

USER
: Your login name.

  Default: $LOGNAME

VISUAL
: The editor you want to use.

Default: $EDITOR

FILES

$DOTDIR/.article
~/dead.article

SEE ALSO

rn(1), Rnmail(1), inews(1)

BUGS

Not the speediest program in the world, but maybe that's a blessing to the net.

NAME

Rnmail - a program for replying via mail

SYNOPSIS

Rnmail destination_list
or
Rnmail -h headerfile [oldarticle]
or
Rnmail

DESCRIPTION

Rnmail is a friendly interface for mailing replies to news articles. It will ask several questions, then allow you to enter your letter, and then mail it off. If you type h and a carriage return at any point, *Rnmail* will tell you what it wants to know.

The -h form is used when invoked from *rn*. If your editor can edit multiple files, and you want the article to which you are replying to show up as an alternate file, define the environment variable MAILPOSTER as "Rnmail -h %h %A". You can also modify the the MAILHEADER environment variable to change the header file that *rn* passes to Rnmail.

ENVIRONMENT

DOTDIR
If defined, specifies a place other than your home directory where 'dot' files may be stored. This is primarily for accounts which are shared by more than one person.

Default: $HOME

EDITOR
The editor you want to use, if VISUAL is undefined.

Default: whatever your news administrator installed, usually vi.

HOME Your home directory.

Default: $LOGDIR

LOGDIR
Your home directory if HOME is undefined.

LOGNAME
Your login name, if USER is undefined.

Default: value of "whoami".

MAILRECORD
If defined, contains the name of a file to which the finished message will be appended.

Default: message not saved

ORGANIZATION
Either the name of your organization, or the name of a file containing the name of your organization.

Default: whatever your news administrator chose.

USER Your login name.

Default: $LOGNAME

VISUAL
The editor you want to use.

Default: $EDITOR

FILES

$DOTDIR/.letter
~/dead.letter

SEE ALSO

rn(1), Pnews(1), mail(1)

BUGS

Uses /bin/mail in the absence of sendmail.

NAME

newsetup - a program to set up a .newsrc file

SYNOPSIS

newsetup

DESCRIPTION

The *newsetup* program creates a new .newsrc file containing all of the currently active newsgroups. It tries to put them in a reasonable order, i.e. local newsgroups earlier, but you'll probably want to change the ordering anyway (if you use *rn*) in order to put interesting newsgroups first. If you already have a .newsrc, it will be backed up with the name ".oldnewsrc".

ENVIRONMENT

DOTDIR
: Where to put your .newsrc, if not in your home directory.

  Default: $HOME

HOME
: Your home directory.

  Default: $LOGDIR

LOGDIR
: Your home directory if HOME is undefined.

FILES

/usr/lib/news/active or a reasonable facsimile
${DOTDIR-{$HOME-$LOGDIR}}/.newsrc

SEE ALSO

rn(1), newsrc(5)

NAME

newsgroups - a program to list unsubscribed newsgroups.

SYNOPSIS

newsgroups pattern flag

DESCRIPTION

The *newsgroups* program compares your .newsrc file with the file of active newsgroups, and prints a list of unsubscribed newsgroups matching pattern. If the second argument "flag" is present, only newsgroups not found in your .newsrc are listed, and the display is not paged. If the second argument is missing, the display is paged, and an additional list of unsubscribed newsgroups occurring in your .newsrc is printed.

ENVIRONMENT

DOTDIR

Where to find your .newsrc, if not in your home directory.

Default: $HOME

HOME Your home directory.

Default: $LOGDIR

LOGDIR

Your home directory if HOME is undefined.

FILES

/usr/lib/news/active or a reasonable facsimile
${DOTDIR-{$HOME-$LOGDIR}}/.newsrc

SEE ALSO

rn(1), newsrc(5)

BUGS

The flag argument is a kludge.

SPMS
SOFTWARE PROJECT MANAGEMENT SYSTEM

Peter Nicklin, Berkeley

The SPMS Software Project Management System

Peter J. Nicklin

Division of Structural Engineering and Structural Mechanics
Department of Civil Engineering
University of California, Berkeley
Berkeley, California 94720

ABSTRACT

The Software Project Management System (SPMS) is a system for the management of medium- to large-scale software systems. SPMS provides, within the UNIX† environment, a number of commands which can greatly simplify many tasks associated with program development and maintenance. SPMS does not attempt to duplicate existing UNIX program development tools such as *make* or *SCCS,* but instead provides a way of coordinating these tools.

SPMS can be fitted to existing software systems. It retains the full capabilities of the UNIX environment with unrestricted access to UNIX tools. As a result, software packages developed using SPMS do not depend on the system for their survival and can be ported to versions of UNIX that do not support SPMS.

July 3, 1985

† UNIX is a trademark of Bell Laboratories.

Table of Contents

1. Introduction 1
2. Simple Tasks 1
2.1. Getting Started 1
2.2. Building a Project 2
2.3. Displaying a Project 3
2.4. Moving Around Inside a Project 4
2.5. Compiling a Program 4
2.6. Moving Files Within a Project 5
2.7. More on Building a Project 5
2.8. Creating a Program Library 6
2.9. More on Developing a Program 6
2.9.1. Included files 7
2.9.2. Program libraries 7
2.9.3. Installation 7
2.10. Global Operations 8
2.10.1. Directory selection 8
2.10.2. Directory order 8
2.11. Locating Files in a Project 9
2.12. Searching Files for Patterns 10
2.13. Changing the Working Project 11
3. Advanced Use 11
3.1. Project Hierarchies 11
3.1.1. The root project 12
3.1.2. Project Pathnames 12
3.2. Project Environment 13
3.3. Global Operations 14
3.3.1. Boolean type label expressions 15
3.3.2. Searching and editing 16
3.4. Testing 16
3.5. Documentation 16
3.5.1. The project log 16
3.5.2. Reference manual 17
3.5.3. On-line help 17
4. Software Management 19
4.1. Program Development Techniques 19
4.1.1. Program compilation 19
4.1.2. Installation 20

4.1.3. Updating 20
4.1.4. Dependency analysis 21
4.1.5. Program checking 21
4.1.6. Version control and releases 21
4.1.7. Function tagging 22
4.1.8. Printing 22
4.1.9. Cleaning up 22
4.1.10. Testing 22
4.1.11. Compound commands 23
4.1.12. User-defined commands 23
4.2. Layered Construction of Software Packages 23
4.3. Maintenance of Software Packages 25
4.3.1. Counting of source lines 25
4.3.2. Cataloging of functions 25
4.3.3. Printing 25
4.3.4. Program checking 26
4.3.5. Testing 26
4.3.6. Comparing versions 26
4.3.7. Releases 26
4.3.8. Cleaning up 27
5. Retrofitting of Software Projects 27
6. Acknowledgements 28
7. References 28
Appendix A. Standard Makefile Templates 30
Appendix B. Project 'vs' Makefile Templates 32
Appendix C. Project Pathname Syntax 36
Appendix D. SPMS Command Summary 37

The SPMS Software Project Management System

Peter J. Nicklin

Division of Structural Engineering and Structural Mechanics
Department of Civil Engineering
University of California, Berkeley
Berkeley, California 94720

1. Introduction

Software packages on the UNIX operating system are frequently organized in a haphazard manner. The conventions for arranging parts of the package within the file system vary from package to package, and absolute pathnames are often used to describe the location of files. The net result is an amorphous non-portable software package requiring a substantial maintenance effort.

It would seem extremely desirable to develop tools and file system structures to support more coherent and portable software packages, and reduce the maintenance effort associated with them. The Software Project Management System (SPMS) is a system for the management of medium- to large-scale software systems. SPMS provides, within the UNIX environment [5], a number of commands which can greatly simplify many tasks associated with program development and maintenance. SPMS does not attempt to duplicate existing UNIX program development tools such as *make* or *SCCS,* but instead provides a way of coordinating these tools.

If only the simpler commands are used, the SPMS system can be helpful for inexperienced UNIX users. If the more advanced SPMS features are used, the experienced user can perform complex tasks with less effort and greater reliability than by applying the standard UNIX tools directly.

Each software package managed by SPMS is organized as a project [7,8]. A project is a collection of directories, each of which supports a specific activity such as program development, testing, or documentation (see fig. 1). There is no restriction on the number of directories belonging to a project. The directory layout is arbitrary, and can be altered to reflect the changing needs of the package.

SPMS can be fitted to existing software systems. It retains the full capabilities of the UNIX environment with unrestricted access to UNIX tools. As a result, software packages developed using SPMS do not depend on the system for their survival and can be ported to versions of UNIX that do not support SPMS.

2. Simple Tasks

In this document several examples related to an interactive screen-oriented spreadsheet program are presented to demonstrate the use of SPMS for software development and project management. It is assumed that the reader is familiar with the UNIX operating system and a text editor such as *ex*. In these examples, user input is shown in **bold** face.

2.1. Getting Started

Before using SPMS for the first time the following steps must be performed[1]

[1] For C shell, (*csh*), users only. Consult the UNIX Programmer's Manual for instructions on how to set up SPMS for the Bourne shell, (*sh*).

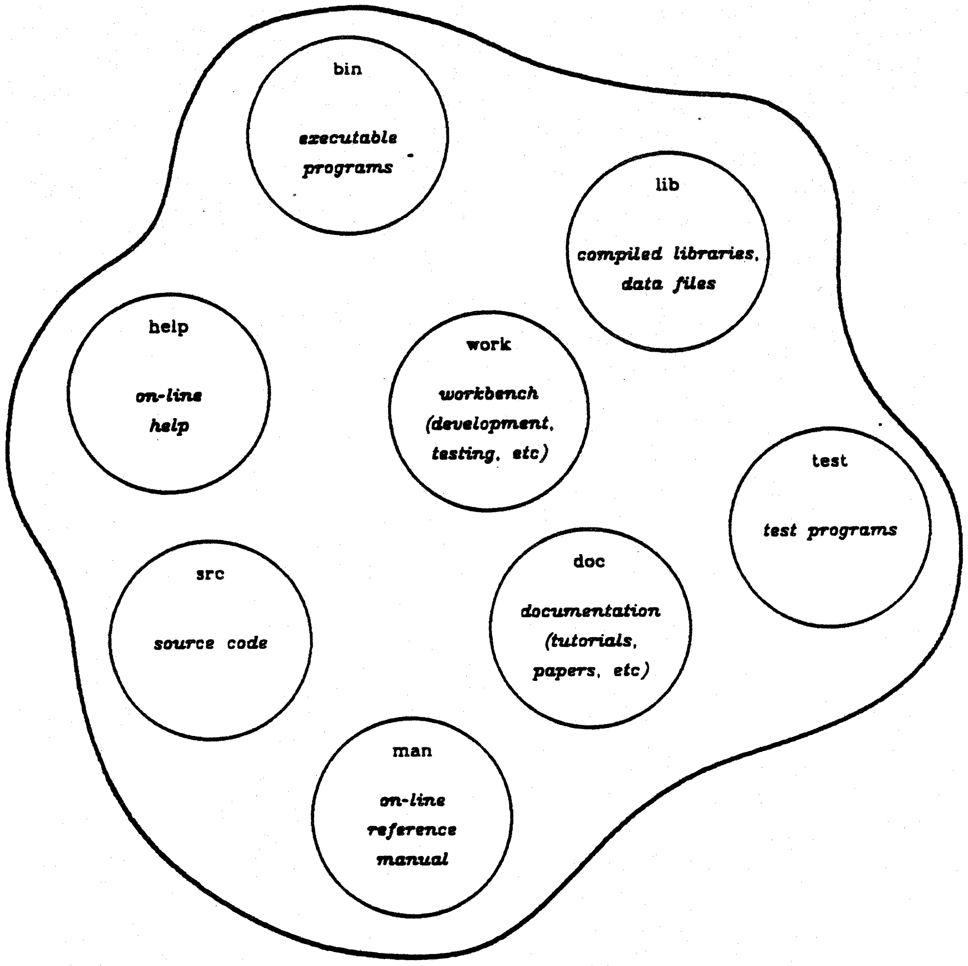


Figure 1. Project organization

1. Include the directory '/usr/new' in the command search path. This is done by altering the PATH environment variable in one of the startup files, '.cshrc' or '.login', in the home directory.
2. Add the following aliases to the '.cshrc' file located in the home directory
```
alias chproject 'eval `"chproject" \!*`'
alias pd 'eval `"pd" \!*`'
```
3. Add the following command to the '.login' file located in the home directory
```
chproject
```
4. Convert the home directory to a project root directory by typing
```
/usr/new/mkproject -d
```
5. Execute the '.cshrc' and '.login' files by typing
```
source .cshrc
source .login
```

2.2. Building a Project

The directory structure to support a software package is created by the *mkproject* and *pmkdir* commands. These commands create directories using the standard UNIX *mkdir* command, and record information about each directory in a project database called the *project link directory*.

This information is used by various SPMS commands to control the development and maintenance activities for a project.

The steps for building the project structure are:

1. Initialize the project using the *mkproject* command. *Mkproject* creates a directory known as a *project root directory,* to serve as the focus for a project, and initializes the project database. After *mkproject* creates the project root directory, the user is prompted for a line describing the purpose of the project.
2. Use the *chproject* command to change to the root directory of the new project and make it the *working project* (see § 2.13).
3. Create the project directories using the *pmkdir* command. After *pmkdir* creates each directory, the user is prompted for a line describing the purpose of the directory.

To illustrate this process, the following commands create project 'vs' with directories 'doc', 'src', and 'work' (see fig. 2) to support a 'Visual Spreadsheet' program[2] called *vs*.

```
% mkproject vs
vs: description? (1 line): Visual Spreadsheet
% chproject vs
% pmkdir doc src work
doc: description? (1 line): vs user's guide
src: description? (1 line): vs program source code
work: description? (1 line): vs workbench
%
```

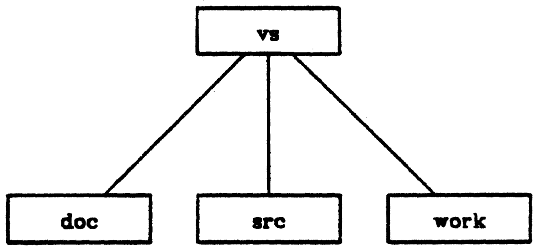


Figure 2. Layout of the project 'vs'

2.3. Displaying a Project

The *ppd* "print project directory" command may be used to list the directories belonging to 'vs':

```
% ppd
doc         src         work
%
```

Alternatively, a table of contents for the project can be obtained by using *ppd* with the –d description option to print the description of each project directory.

[2] *Vs* is a fictitious name bearing no resemblance to any actual program.

```
% ppd -d
doc         vs user's guide
src         vs program source code
work vs workbench
%
```

2.4. Moving Around Inside a Project

The *pd* command provides a convenient way for changing to another project directory without the user having to remember it's precise location. For example, to move to the source code directory 'src', type

```
% pd src
```

To change to the directory 'work', type

```
% pd work
```

To return to the project root directory, type

```
% pd
```

without any arguments.

2.5. Compiling a Program

Program development and maintenance is handled by the *make* command[3]. *Make* mechanizes many development and maintenance activities, including compiling and linking of programs, printing of source code, and the removal of unneeded files. The instructions which tell *make* how to perform these duties are kept in a special file known as a makefile, together with the names of the source code files which make up the program. The makefile editor program, *mkmf,* creates the makefile (named 'Makefile' by default) by gathering up the names of all the source code files in the current working directory and inserting them into a standard makefile.

The following example shows how to produce the program for the visual spreadsheet, given the file 'vs.c' containing the source code in the directory 'src'.

```
% mkmf
mkmf: creating Makefile from template /usr/new/lib/p.Makefile
% make
cc -c vs.c
Loading a.out ... done
%
```

In this example the executable program is called 'a.out'. However, by using the makefile editor interactively the name 'vs' could have been specified instead:

```
% mkmf -i
mkmf: creating Makefile from template /usr/new/lib/p.Makefile
program name? vs
destination directory?
% make
cc -c vs.c
Loading vs ... done
%
```

Since a carriage return was typed in response to the second question in the example above, the destination directory for the program remains the current directory.

Because program *vs* is a screen-oriented program, it would not be surprising if it requires special functions to control cursor movement and updating of the terminal screen. There is a standard package of C library functions for this purpose called 'curses' [1], and if the program has taken advantage of these functions, this library should be included in the makefile together with the terminal database package 'termlib'. This can be done by including the LIBS macro definition as an argument to the *mkmf* command[3]

```
% mkmf -i "LIBS=-lcurses -ltermlib"
```

2.6. Moving Files Within a Project

A file can be moved to another project directory by using the *pmv* command. For instance, the following command moves the executable program *vs* from the current working directory to the 'work' directory

```
% pmv vs work
```

In a similar manner, files can be copied from one project directory to another using the *pcp* command. *Pmv* and *pcp* behave very similarly to the standard UNIX *mv* and *cp* commands in that they blindly overwrite any existing files of the same name in the destination directory unless the **-i** interactive option is used.

2.7. More on Building a Project

As development of a software package continues, extra project directories may be needed to support the work. For example, project 'vs' must accommodate an additional program called *vstutor* which provides instruction on the use of the visual spreadsheet program; two library packages called 'hash' and 'list' for hash table and linked list operations; and three files that are "included" in more than one source file – 'vs.h' which contains common program definitions, 'hash.h' which defines hash tables, and 'list.h' which holds linked list definitions. Figure 3 shows the extra directories needed for these components and the following command sequence creates them

```
% pd
% pmkdir bin include lib
bin: description? (1 line): vs and vstutor programs
include: description? (1 line): common included files
lib: description? (1 line): compiled hash table and list libraries
% pd src
% pmkdir vs vstutor libhash liblist
vs: description? (1 line): vs program source code
vstutor: description? (1 line): vstutor program source code
libhash: description? (1 line): hash table library source code
liblist: description? (1 line): list library source code
%
```

The final step is to change the description of the 'src' directory now that it has been subdivided into four separate source code directories. This can be done by using the *pmkdir* with the +d (change description) option

```
% pmkdir +d src
src: description? (1 line): C source code
%
```

Note that in Figure 3 there are two directories called 'vs'. The top one bears the name of the project, and the bottom one is named according to the program contained within it. Similarly, the directories 'libhash' and 'liblist' are named according to libraries that they contain.

[3] Arguments with embedded blanks in UNIX commands must be enclosed by double quotes.

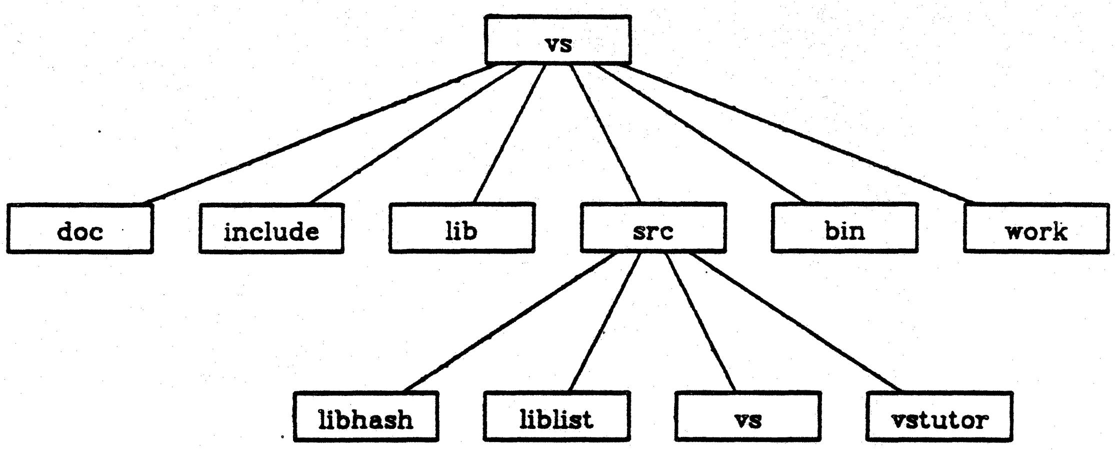


Figure 3. Revised layout of project 'vs'

2.8. Creating a Program Library

A program library is a collection of compiled subroutines that are shared by more than one program. In the UNIX environment a program library is stored as an *archive* file. Each member of the archive is an object file containing one or more compiled subroutines. By convention a library archive file is named **lib***name*.**a** where *name* is the name of the program library.

The example below shows how to create a program library for the hash table subroutines in the 'libhash' directory. Note that the *mkmf* command must be given with the –l option so that a makefile will be created for a library rather than a program.

```
% mkmf -i -l
mkmf: creating Makefile from template /usr/new/lib/l.Makefile
library name? libhash.a
destination directory? ../../lib
% make
cc -c hthash.c
cc -c htinit.c
cc -c htinstall.c
cc -c htlookup.c
cc -c htrm.c
Loading libhash.a ... done
%
```

Since the 'lib' directory is in the same project as the 'libhash' directory, the path to 'lib' is made *relative* to 'libhash' so that the project will be portable.

The next step is to install the program library in the 'lib' project directory where the *vs* and *vstutor* programs can access it easily.

```
% make install
Installing libhash.a in ../../lib
%
```

2.9. More on Developing a Program

2.9.1. *Included files*

Definitions which are common to more than one source code file (e.g. buffer sizes, data structure definitions) should be declared only once in a program. This can be achieved by keeping such definitions in files separate from the main program and "including" them at compilation time. In C, Fortran, and Pascal programs, the contents of a file can be included by the statement

```
#include "filename"
```

By convention *filename* ends in **.h** and is commonly referred to as a *header* file. Hence, in the source code for programs *vs* and *vstutor,* the statements

```
#include "vs.h"
#include "hash.h"
#include "list.h"
```

include common program definitions, hash table definitions, and linked list definitions respectively.

Since the header files in this example are used in more than one program, they should be placed in the 'include' directory where they can be accessed easily. Although the include statements can be rewritten as

```
#include "../../include/vs.h"
#include "../../include/hash.h"
#include "../../include/list.h"
```

it is better to tell the compiler where the header files are by using the –I compiler option[4] as follows

```
-I../../include
```

This is done most conveniently by adding the option to the compiler flags in the makefile (see § 4.1.1).

2.9.2. *Program libraries*

The LIBS macro definition in a makefile specifies the libraries that are to be used by the link editor for resolving references to subroutines that are not found in the program source code. Because *make* checks to see if the libraries needed by a program have changed since the last time the program was made, their pathnames must be defined explicitly. In the makefiles belonging to programs *vs* and *vstutor,* the LIBS macro definition looks like

```
LIBS = ../../lib/libhash.a \
       ../../lib/liblist.a \
       /usr/lib/libcurses.a \
       /usr/lib/libtermlib.a
```

Note also that when this macro definition was added to the makefile by the command

```
% mkmf "LIBS=../../lib/libhash.a ../../lib/liblist.a -lcurses -ltermlib"
```

to include the 'hash' and 'list' libraries, the 'curses' and 'termlib' libraries were automatically expanded to full pathnames by the makefile editor.

2.9.3. *Installation*

Once a program has been completed, it should be installed in a place where it will be generally available – that is, in a directory which is in the command search path specified by the PATH environment variable. In the case of the project 'vs', if the 'bin' directory is in the search path, this might be a good place to install the *vs* and *vstutor* programs. If the makefiles for these

[4] C and Fortran compilers only.

programs do not already specify 'bin' as their destination directory, it can be added by the command

```
% mkmf "DEST=../../bin"
```

Then, each program can be installed by the *make install* command. For the program *vs*:

```
% pd vs
% make install
Installing vs in ../../bin
%
```

and for the program *vstutor*:

```
% pd vstutor
% make install
Installing vstutor in ../../bin
%
```

2.10. Global Operations

One of the goals of SPMS is to reduce the effort associated with software maintenance. This can be achieved by treating a software package as an atomic unit – that is, a single entity on which to perform operations. The mechanism for executing a command over an entire software package is provided by the *pexec* command. This command takes another command as an argument and executes it in each of the directories belonging to a project, as in

```
% pexec ls
```

which lists the names of all the files in a project.

2.10.1. *Directory selection*

By labeling each project directory according to the type of activity that it supports, global operations can be restricted to specific directories. These labels, which are known as *type labels,* are attached to project directories by the *pmkdir* command, and removed by the *prmdir* command[5]. For instance, if the directories containing source code in project 'vs' are labeled 'src' by

```
% pmkdir +T src include libhash liblist vs vstutor
```

then, the total number of lines of source code in a project can be counted by giving the command

```
% pexec –T src 'cat *.h *.c' | wc –l
```

where quotes surround the *cat* command to prevent file name expansion in the current directory.

If a project directory supports more than one type of activity, labels corresponding to each of the activities can be attached to the directory.

2.10.2. *Directory order*

In some instances the directories affected by a global command must be processed in a particular order. For example, when installing a software package which has both libraries and programs, the libraries should be installed first. This ordering is achieved by appending priorities to type labels. In the case of the project 'vs', if the directories containing the program and library source code are labeled 'install' with the following priorities

[5] Except in the case of project root directories, where *mkproject* and *rmproject* must be used.

Directory	Priority
libhash	1
liblist	1
vs	2
vstutor	2

by the commands

```
% pmkdir +Tinstall.1 libhash liblist
% pmkdir +Tinstall.2 vs vstutor
```

then, the command

```
% pexec -Tinstall make install
```

installs the 'vs' software package in the order shown in figure 4.

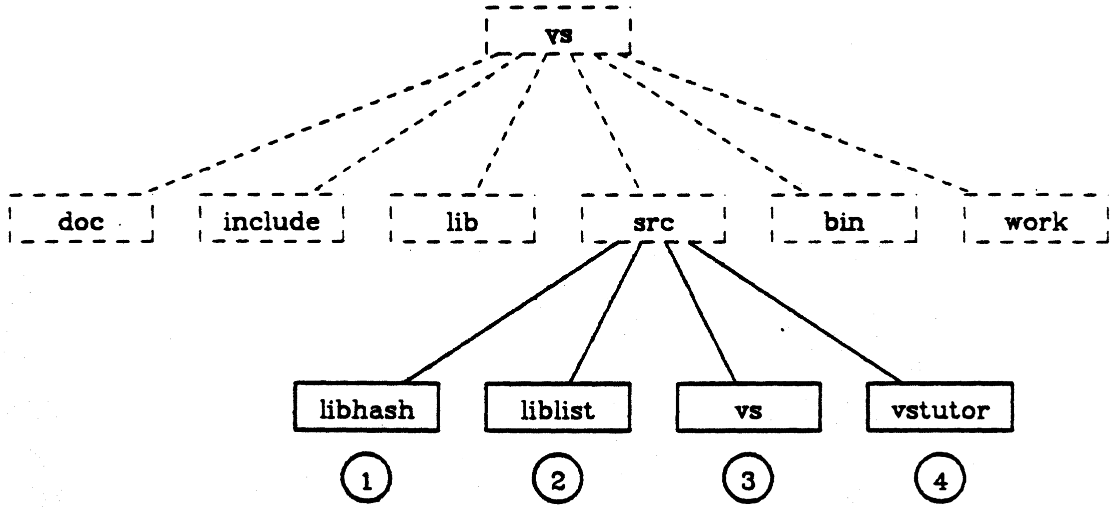


Figure 4. Ordering for 'install' directories

In a similar fashion, if the directories containing source code are labeled 'print' with the following priorities,

Directory	Priority
include	1
vs	2
vstutor	2
libhash	3
liblist	3

a source code listing for the entire project may be obtained by the command

```
% pexec -Tprint 'pr *.h *.c' | lpr
```

in the order shown in figure 5.

2.11. Locating Files in a Project

When the location of a file within a project is unknown, it can be found by using the *pfind* command. For example, the command

```
% pfind Makefile
```

searches for all occurrences of 'Makefile' in project 'vs' and produces the output

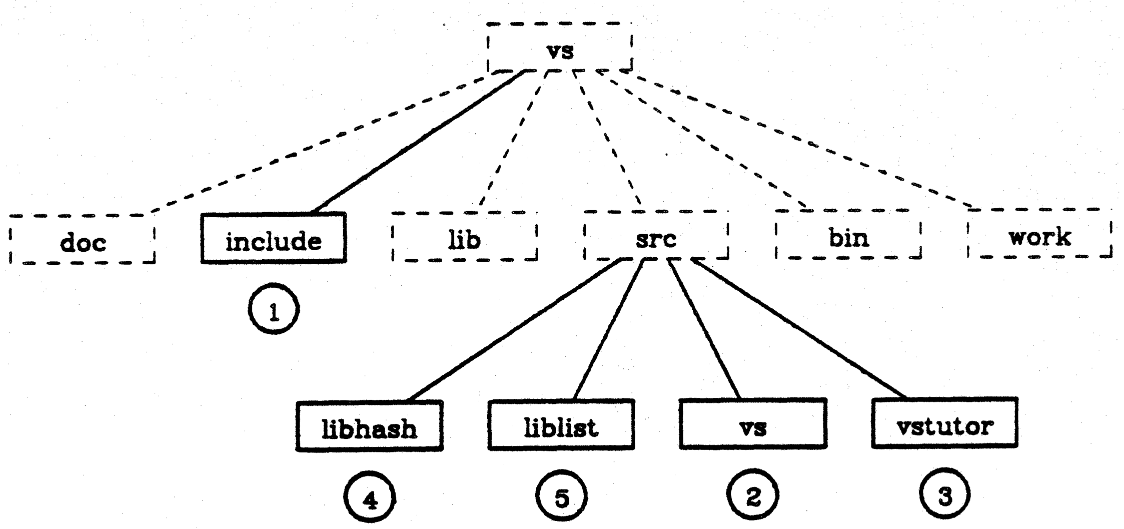


Figure 5. Ordering for 'print' directories

```
...^vs/Makefile
...^vstutor/Makefile
...^libhash/Makefile
...^liblist/Makefile
```

In a large project, the time required to search for a file can be reduced by telling *pfind* to scan only those directories in which there is some likelihood of the file being found. In the example above, since makefiles are only likely to be found in source code directories (i.e. directories having type label 'src'), the command could have been given as

```
% pfind -Tsrc Makefile
```

2.12. Searching Files for Patterns

Sometimes it is necessary to look at all the files in a software package that contain a certain pattern[6]. One reason might be to find all of the places from which a subroutine is called, perhaps with the intent of altering its arguments. The *pgrep* command searchs through specified files in a project for lines matching a given pattern. For example,

```
% pgrep -Tsrc listappend '*.h *.c'
```

will search all the C source code files in project 'vs' for the function 'listappend'. Because of the **-T** option, *pgrep* searchs only in those directories which have the 'src' type label.

An alternative way for specifying file names is to use the **-m** option. This causes *pgrep* to fetch the names of source code files from the HDRS and SRCS macro definitions in a makefile. Consequently, the command in the example above could have been expressed as

```
% pgrep -Tsrc -m listappend
```

If the pattern contains characters that have a special meaning to the shell, such as * or ^, the pattern should be quoted. For example,

```
% pgrep -Tsrc -m 'ht.*('
```

finds all of the places where functions from the hash table library are used.

[6] The term *pattern* is used to denote a set of strings.

2.13. Changing the Working Project

Along with a working directory, each user has a *working project*. Immediately after logging in, the working project is the root project '^'. To change to a new working project, the *chproject* command must be used, as in

```
% chproject vs
```

which makes 'vs' the current (or working) project. To return to the root project, execute the command

```
% chproject ^
```

To find out the name of the working project, type

```
% pwp
```

3. Advanced Use

This section summarizes the rest of the facilities offered by SPMS for handling large software projects. Techniques for searching and editing text files, program testing, and documentation are explained.

3.1. Project Hierarchies

To facilitate the management of large projects, such projects can be subdivided into smaller projects. These subprojects can be nested to any level to form a project hierarchy which is very similar to the UNIX directory hierarchy. For example, as project 'vs' grows, it might be convenient to convert each of the 'hash' and 'list' program libraries into subprojects (see fig. 6).

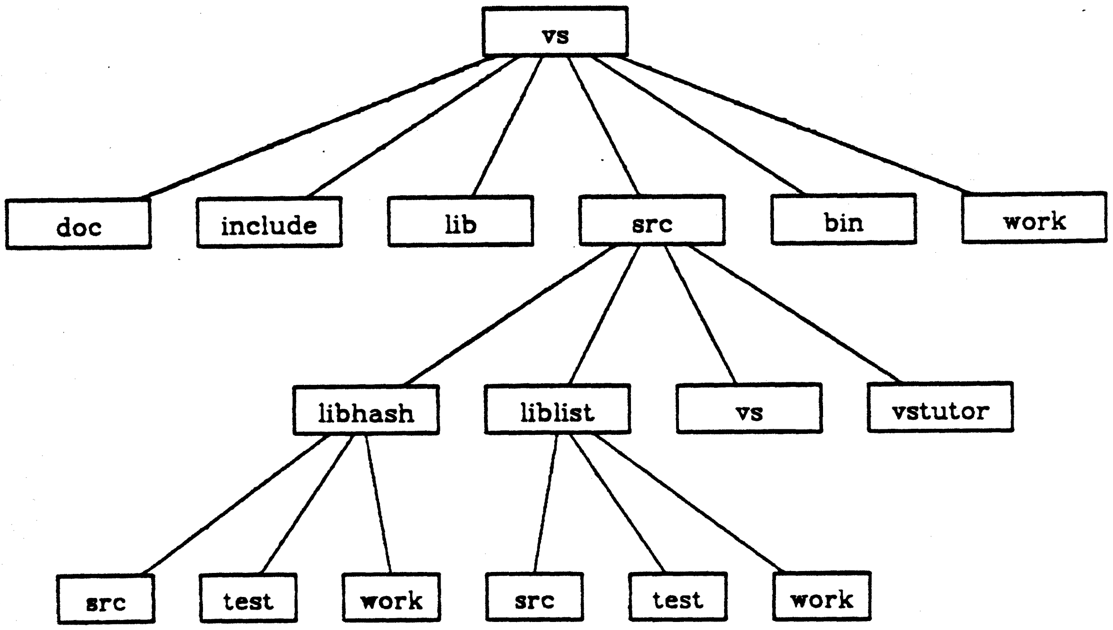


Figure 6. Project 'vs' with subprojects

To show how this conversion is done, the following set of commands converts project directory 'libhash' into a subproject.

Convert the project directory into a subproject . . .

```
% pd src
% prmdir -u libhash
% mkproject libhash
libhash: description? (1 line): VS Hash Table Operations
```

. . . create the project directories . . .

```
% chproject libhash
% pmkdir src test work
src: description? (1 line): hash table library source code
test: description? (1 line): hash table library test programs
work: description? (1 line): hash table library workbench
```

. . . reattach the type labels . . .

```
% pmkdir +T libsrc,install.1,print.3  src
```

. . . and rearrange the library

```
% pmv Makefile *.c *.o src
% pd src
% mkmf DEST=../../../lib
```

3.1.1. *The* **root** *project*

The project at the top of each user's project hierarchy is called the *root project* and is given the special name '^'. When the SPMS system was initially set up (see §2.1), the command

```
% mkproject -d ^
```

created the root project and made the user's home directory into the project root directory for '^'.

3.1.2. Project Pathnames

Project pathnames provide a convenient way for accessing a particular directory or file within a project hierarchy[7]. A project pathname is formed by a succession of project names separated by '^' characters[8], followed by the name of the directory or file. For instance, the pathname

```
^vs^libhash^src
```

represents the path from the root project to the 'src' directory located in subproject 'libhash', and the pathname

```
^vs^libhash^src/hthash.c
```

locates the file 'hthash.c' in that directory.

A project pathname can be *absolute* or *relative.* An *absolute project pathname* specifies the path from the root project and begins with the character '^'. However, a project pathname not beginning with '^' is interpreted with respect to the current working project and is therefore called a *relative project pathname.* For example, the pathname

```
libhash^src
```

specifies the location of 'src' relative to project 'vs', assuming that 'vs' is the working project.

[7] Project pathnames are recognized only by SPMS commands.

[8] Since the Bourne shell, (*sh*), recognizes the '^' character as an alternative pipe symbol, Bourne shell users must type '\^' instead.

Since relative project pathnames are interpreted relative to the current working **project** rather than the current working directory, this means that project directories and files can be accessed from **any** working directory. For example, the command

```
% pmv src/libhash.a work
```

moves the hash table library from the 'src' directory in the working project 'libhash' to the 'work' directory in the same project, regardless of the location of the current working directory.

The parent of the working project is called '....' and may be used in a project pathname to go up one level in a project hierarchy. Thus, the command

```
% chproject ....
```

makes the parent project of the current project into the new working project. If the current project happens to be 'libhash', then the command

```
% chproject ....^liblist
```

will change to project 'liblist'. For completeness, '...' is an alternative name for the current working project. Table 1 summarizes the conventions used in project pathnames together with the equivalent conventions for regular pathnames.

Table 1. Pathname conventions

Project Pathname		Regular Pathname	
Component	Description	Component	Description
^	root project	/	root directory
^	separator character	/	separator character
...	working project	.	working directory
....	parent project	..	parent directory

Project pathnames can be modified in two ways. The first way allows a user to refer to a project belonging to someone else by prepending ~*user* to the pathname. For example, if 'root' has a copy of the project 'vs', the command

```
% ppd ~root^vs
```

will print the directories in that project. The other way allows a regular pathname to follow a project pathname, separated by a '/' character. This enables access to directories which are not part of a project. To illustrate, if 'junk' is a regular directory in the 'work' directory of the project 'vs', the command

```
% pd ^vs^work/junk
```

changes to that directory.

3.2. Project Environment

It is possible to tailor the environment for the current project by adding commands to the '.projectrc' startup file located in the root directory of the project. When the project is activated by the *chproject* command, this file is executed. For instance, if a user wishes to be reminded of tasks that still need attention on a project, a reminder service can be set up by putting the reminders in a file, (e.g. '.reminder') and adding the line

```
cat .reminder
```

to the '.projectrc' file.

It is also a good idea to include the **-d** option in the alias for the *chproject* command (see § 2.1) so that when *chproject* is invoked, it will print the name of the new working project, as in

```
% chproject ˆvs
Visual Spreadsheet
%
```

3.3. Global Operations

Even if a project is divided into subprojects, commands can still be executed globally over the entire software package by the *pexec* command. *Pexec* has two modes of execution, depending on the method chosen for selecting directories. If type labels are not used for selecting a particular set of directories, *pexec* descends recursively through the project hierarchy and executes the command argument in the project directories at each level. The command

```
% pexec ls
```

demonstrates this mode of operation by listing the contents of the directories in the project 'vs' in the order shown in figure 7.

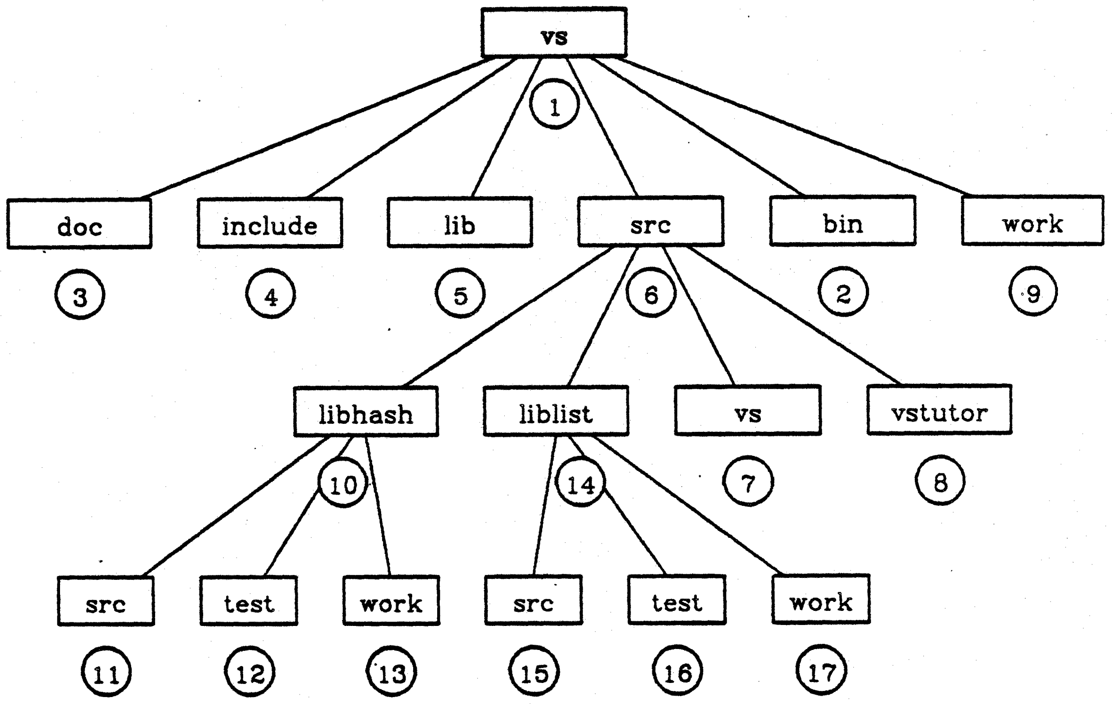


Figure 7. Directory ordering for 'pexec ls'

The other mode of operation, involving the use of type labels, causes *pexec* to search the project hierarchy for directories with appropriate type labels, sort the directories according to their priorities, and then execute the command argument in each directory. As an example of this mode of execution, figure 8 indicates the order in which the command

```
% pexec -Tprint 'pr *.h *.c' | lpr
```

prints the project 'vs'.

With both modes of operation, *pexec* resets the current working project to the project in which the directory resides. For each of the 'src' directories in project 'vs' the corresponding working projects are

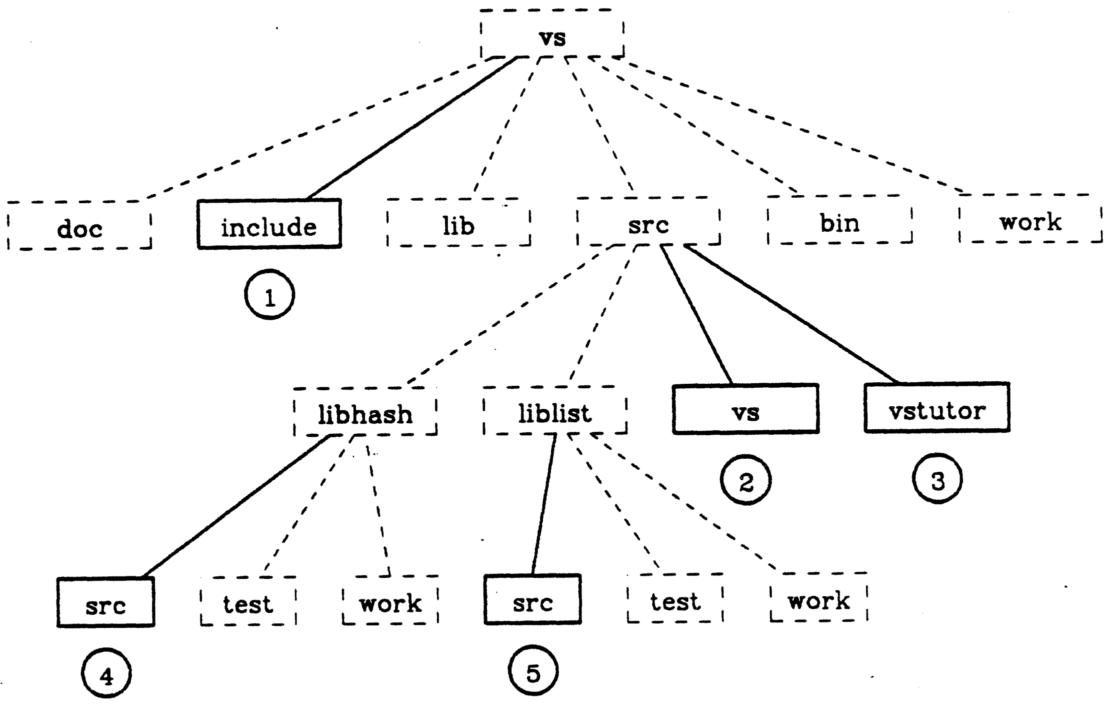


Figure 8. Directory ordering for 'pexec -T print ...'

Directory	Working Project
^vs^src	^vs
^vs^libhash^src	^vs^libhash
^vs^liblist^src	^vs^liblist

3.3.1. *Boolean type label expressions*

Global commands can be made even more precise by using boolean expressions[9] on type labels to select project directories. To show how boolean expressions are used, let the source code directories in project 'vs' have the type labels shown below.

Directory	Type Labels
^vs^include	print.1, include, src
^vs^vs	print.2, install.2, cmdsrc, src
^vs^vstutor	print.2, install.2, cmdsrc, src
^vs^libhash^src	print.3, install.1, libsrc, src
^vs^liblist^src	print.3, install.1, libsrc, src

In terms of entering the boolean expression on the command line, '**or**' is represented by the character '|', '**and**' by the character '&', and '**not**' by '!'. Since these characters, together with '(' and ')', are meaningful to the command shell, it is good idea to enclose the whole expression in quotes[10]. Then, the command

[9] The formal definition of a boolean type label expression is

$$E \rightarrow E \text{ or } E \mid E \text{ and } E \mid \text{not } E \mid (E) \mid \text{id}$$

where E is a boolean expression; **and**, **or**, and **not** are boolean operators; and **id** is a type label. As is customary, it is assumed that **or** and **and** are left-associative, and that **or** has the lowest precedence, then **and**, then **not**.

[10] Even if this is done, the '!' character must still be escaped by a backslash '\' if it precedes a type label to

```
% pexec "-Tprint&(libsrc|cmdsrc)" 'pr *.h *.c' | lpr
```

prints the source code in both the program and library source code directories, but not the directory containing header files. Alternatively,

```
% pexec "-Tprint&\!include" 'pr *.h *.c' | lpr
```

achieves the same result.

3.3.2. *Searching and editing*

Whenever it becomes necessary to alter something like the number of arguments in a call to a function, the *pgrep* command can be used to bring up the text editor on all the files in the software package that contain that function call. Suppose, for example, that the number of arguments to the UNIX system call 'open' for opening files has changed. The command

```
% pgrep -Cvi -Tsrc -m 'open('
```

will edit all the source code files containing that call, using the *vi* text editor.

3.4. Testing

After a program is released for general use, it will require maintenance. It may have to be modified to speed it up, fix bugs, or add new features. Each time the program is altered, the parts that are affected should be checked against previous test results by doing *regression* testing. The *ptest* program mechanizes this process.

Ptest tests each function by running a test program and comparing the output with previously prepared results. For example, the test for the 'htinstall' function in the hash table library produces

```
% ptest htinstall
htinstall: extracting archive ... compiling test ... executing test ... done
%
```

if the test succeeds. However, if the test fails, *ptest* reports this fact by

```
htinstall: extracting archive ... compiling test ... executing test ... failed
```

and saves the error diagnostics in a file named 'Ehtinstall'.

The test program and data files for each test case are stored in an archive file named *test*.a where *test* is the name of the test case, located in the 'test' directory. In the case of 'htinstall', the test archive is called 'htinstall.a' and contains the test program source file, Thtinstall.c, the input data file, Ihtinstall, and the validated output data file, Ohtinstall. The details on how to set up a test archive are explained more fully in section ptest(1P) of the UNIX programmer's manual.

3.5. Documentation

3.5.1. *The project log*

The *plog* project log command provides an electronic notebook system by which to record transactions such as incoming and outgoing mail, progress reports, minutes of project staff meetings, etc. *Plog* records messages in a file called 'projectlog' located in the project root directory, by invoking the UNIX *Mail* program [9]. After the *Mail* program starts up, the user types in the message, followed by a period '.' or CNTL-D at the beginning of a line. Since the *Mail* program processes the message, the user can take advantage of all the mailing facilities offered by the system. For instance, the following announcement on the 'vs' project can be mailed to a group of users labeled 'vsusers'[11] using the '~c' 'carbon copy' facility of the *Mail* program:

prevent it from being interpreted by the *csh* history mechanism.

[11] By the *alias* mechanism of *Mail*.

```
% plog
Subject: 'vs' release 2
~c vsusers
Release 2 of the 'vs' visual spreadsheet package is now available for
distribution. It has the following features:
                    .
                    .
                    .
%
```

Plog can be used to produce reports by printing sections of the project log with subject headings. For example, if the above announcement is message 20 in the project log for the 'vs' project, the following command will print message 20 plus any subsequent messages.

```
% plog -p20
------------------------------------------------------------------
                          'vs' release 2
------------------------------------------------------------------
From pjn Wed Aug 10 11:02:44 1983
To: /usr/pjn/vs/projectlog
Subject: 'vs' release 2
Cc: vsusers

Release 2 of the 'vs' visual spreadsheet package is now available for
distribution. It has the following features:
                    .
                    .
                    .
%
```

Plog can also be used to collect incoming mail, edit the project log, and sort it into chronological order. These options are explained more fully in section plog(1P) of the UNIX programmer's manual.

3.5.2. *Reference manual*

The *pman* command supports a project reference manual in the same way that the *man* command provides information from the UNIX programmer's manual. For example, to print information about the 'vs' visual spreadsheet program, type

```
% pman vs
```

and to find out about the 'vstutor' program, type

```
% pman vstutor
```

The directories that contain the manual entries must be set up in the same way as the programmer's manual as shown in figure 9. By convention, manual pages for commands have '.1' suffixes and are kept in the 'man1' directory, manual pages for libraries have '.3' suffixes and are kept in 'man3', and file formats have '.5' suffixes and are kept in 'man5'. The *pman* command searchs through each of the 'man1', 'man3', and 'man5' directories in turn until it finds the topic.

3.5.3. *On-line help*

On-line help for a project is provided by the *phelp* command. After *phelp* is typed, it prints some introductory information, a list of available topics, and then the prompt '???', indicating that it is ready for a command. The following commands are recognized

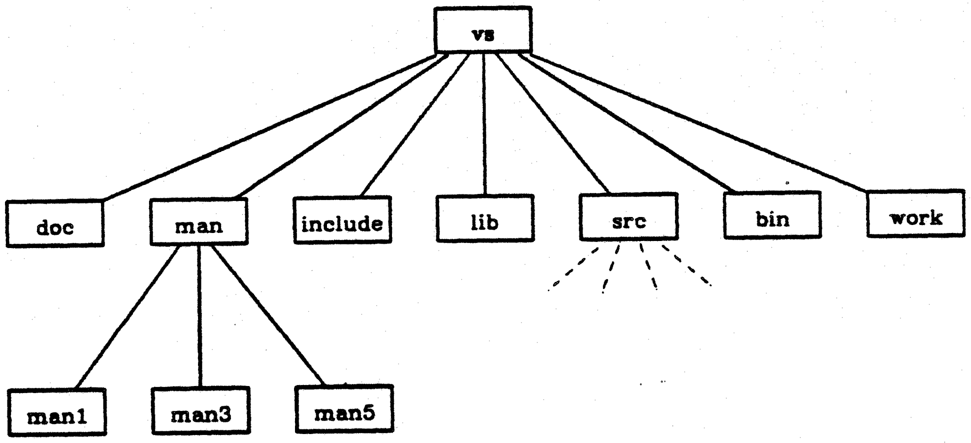


Figure 9. Layout of the 'vs' project manual

Command	Response
a topic name	Print help information on topic.
index	Display list of topics available at this level.
help	Display help on how to use *phelp.*
?	Display this command summary.
q	Exit from *phelp.*
P *projectname*	Change to another project.
~	Return to the top level of help topics.
carriage return only	Go up one level of help topics.

If a topic name is typed in reply, *phelp* will print a page of information and then wait until a space is typed before it continues.

In project 'vs' there are three topics available – 'install' explains how to install 'vs'; 'progress' lists recent developments; and 'schedule' outlines the development plan. In the following session, *phelp* is used to examine some of these topics.

```
% phelp
(prints an introduction to phelp)

Help topics available:  install  progress  schedule

??? schedule
(prints 'schedule' topic)
??? progress
(prints 'progress' and goes down one level to 'progress' subtopics)

progress subtopics: bugfixes

progress-->??? bugfixes
(prints 'bugfixes' topic)
progress-->??? q
(exits from phelp)
%
```

Help topics are contained in files which reside in the 'help' directory located in the project root directory. Figure 10 shows how these topics are set up for project 'vs'.

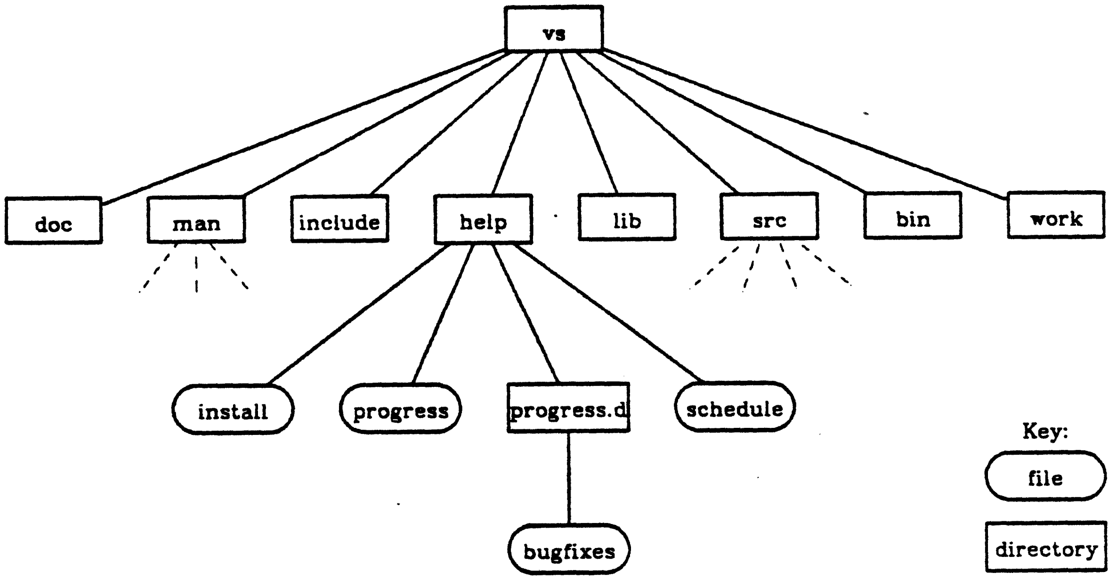


Figure 10. Help topics for project 'vs'

The circles represent topic files, and the rectangles represent directories. Subtopics are contained in subdirectories named according to the topics they represent, but with a '.d' suffix. Consequently, 'bugfixes' is in the subdirectory 'progress.d' since it is a subtopic of 'progress'.

4. Software Management

Although the UNIX operating system offers a rich variety of programming tools, a frequent complaint is that there are too few guidelines showing how to use them in a coherent way. SPMS provides the 'glue' for coordinating the use of these tools and this section describes the techniques which have been devised for the development and maintenance of software packages.

4.1. Program Development Techniques

Before discussing the maintenance of complete software packages, it is worthwhile to review the basic commands for managing individual programs and libraries. The commands are summarized in table 2 and explained below in more detail.

4.1.1. *Program compilation*

The command

```
% make
```

compiles source files into object files and loads them together to produce an executable program. Although *make* uses built-in information for generating the object files from the source files, the decision on how to load the program is left to the user. By default, the program makefiles produced by *mkmf* use the C compiler for this purpose. However, if the programming language is not C, the LINKER macro definition in the makefile should be altered accordingly. For Fortran this can be done by typing the command

```
% mkmf LINKER=f77
```

and for Pascal

Table 2. Program development commands

Task	Command
Program compilation	make
Installation	make install
Updating	make update
Dependency analysis	make depend
Program checking	make lint
Version control and releases	make co
Function tagging	make tags
Printing	make print
Cleaning up	make clean
Testing	ptest

```
% mkmf LINKER=pc
```

Compiler options can be specified by adding certain macros to a makefile. For example, the macro

```
CFLAGS = -O
```

causes a C program to be compiled with optimization, and the macro

```
CFLAGS = -I../../include -g
```

tells the C compiler to search for header files in the directory '../../include' as well as in the current directory (see §2.9.1), and to compile the program with debugging information. FFLAGS and PFLAGS can be used similarly to set options for the Fortran and Pascal compilers respectively.

4.1.2. *Installation*

The command

```
% make install
```

installs a program or library (see §2.9.3). If any source code files are newer than their corresponding object files they are recompiled and the program or library reformed. Even if the object files are up-to-date, a program may still be relinked if the libraries on which it depends are newer than the program itself.

Normally a program is stripped of its symbol table and relocation bits when it is installed, to save space. This can be avoided if the **-s** option is removed from the *install* command line in the makefile.

4.1.3. *Updating*

If a program or library is out-of-date - that is, some of the source code files are newer than the installed version - the command

```
make update
```

recompiles and reinstalls the program or library. This command is more powerful than *make install* because it is not affected by the absence of the object files. In the case of an out-of-date library, all the object files are extracted from the library before any recompilation takes place, and removed once the library has been reinstalled.

4.1.4. *Dependency analysis*

Although the *make* program has a set of built-in rules for recompiling a program if any of the files on which it depends have changed since the last time it was constructed, these rules do not extend to included files. It is necessary to add explicit dependency rules to a makefile so that if a header file is changed, the affected source files will be recompiled and a new program produced. For instance, if the rule

```
vs.o: vs.h  hash.h  list.h
```

is added to the makefile for the 'vs' program, the file 'vs.c' will be recompiled if any of the included files (see § 2.9.1) have changed since the last time it was compiled. The command

```
% make depend
```

calls on the *mkmf* makefile editor to insert header file dependencies into a makefile.

4.1.5. *Program checking*

Programs written in C can be checked for bugs, obscurities, wasteful or error prone constructions, type and function usage, and portability by the *lint* program [4]. If a makefile contains the line

```
lint:;  @lint $(LINTFLAGS) $(SRCS) $(LINTLIST)
```

where the LINTFLAGS macro definition specifies *lint* options, SRCS represents the source files making up the program or library, and LINTLIST is a list of lint libraries (see below), the command

```
% make lint
```

will check that the program or library is consistent.

To set up the 'vs' program might be set up for "linting", the macro definitions would be

```
LINTFLAGS = -I../../include

SRCS      = vs.c

LINTLIST  = ../../lib/llib-lhash.ln \
            ../../lib/llib-llist.ln \
            -lc
```

Just as program libraries share functions among different programs, lint libraries can be used to check that those functions have been used correctly. Lint libraries are created by *lint -C* as shown by the following entry in the makefile belonging to the 'hash' library

```
$(LINTLIB): $(SRCS) $(HDRS) $(EXTHDRS)
            @echo "Loading $(LINTLIB) . . ."
            @lint $(LINTFLAGS) -C$(LIBNAME) $(SRCS)
            @echo done
```

where LIBNAME is defined in the makefile as 'hash', and LINTLIB is defined as 'llib-l$(LIBNAME).ln' in accordance with standard lint library naming conventions.

4.1.6. *Version control and releases*

During the time that a program is being developed it is quite likely that it will undergo several revisions. Features are added and algorithms are improved. Often changes are made which are later found not to work and need to be undone. One way to handle these changes is to save a copy of each file before it is revised. However, this quickly becomes expensive in terms of space. A better solution is to use a version control system like *SCCS* (Source Code Control System) [6] or *RCS* (Revision Control System) [10] which stores only the changes made to the source code together with details such as when each change was made, why it was made, and who made it.

Once a program is ready for release, all of the source files should be stamped with a common version name or number so that it can be recreated at any time regardless of any subsequent changes. *RCS* has the advantage over *SCCS* in this respect because it enables the user to stamp each release with a unique name. For example, if the makefile and all the source files in release 2 of the 'list' library are stamped 'V2', the command sequence

```
% co -rV2 Makefile
% make VERSION=V2 co
```

will create that release by extracting or "checking out" the Makefile and the source files from the *RCS* system using the *co* command.

4.1.7. *Function tagging*

By creating a database of function names[12] with the command

```
% make tags
```

it is possible for the user to find and edit a function without having to remember the name of the file in which the function is located. For example, in the 'src' directory of the 'libhash' subproject, the command

```
% vi -t htinit
```

invokes the *vi* editor on the file 'htinit.c' and positions the cursor at the beginning of the 'htinit' hash table initialization function.

A list of functions which make up the program or library, together with the line number and file in which each is defined, can be obtained by the command

```
% make index
```

4.1.8. *Printing*

To print all of the program source and header files on the line printer *lpr,* type

```
% make print | lpr
```

By default the files are formatted by *pr* so that the output is separated into pages headed by a date, the name of the file, and a page number. Another format can be specified by changing the PRINT macro definition in the makefile.

4.1.9. *Cleaning up*

To save space once a program has been completed, the command

```
% make clean
```

removes object files plus any other files which can be regenerated easily.

4.1.10. *Testing*

Using the test cases prepared previously (see §3.4), it is possible to test an entire program or library by typing the command

```
% ptest
```

Ptest reports the outcome of each test – i.e. whether it passes or fails – and if it fails, saves the error diagnostics in a file called E*test* where *test* is the name of the test.

[12] For C, Fortran, and Pascal programs only.

Because *ptest* uses a number of temporary working files for each test and creates an error diagnostic file for each test that fails, it is a good idea not to clutter up the directories that contain source code or test cases, but instead perform the testing in another directory such as 'work' (see § 4.3.5).

4.1.11. *Compound commands*

The *make* program can process multiple requests. For example,

```
% make install tags clean
```

installs a program or library, creates function tags, and removes any unneeded files. The only operation which may cause problems if it is used in conjunction with *install* or *update* is *make depend* because it recreates the include file dependencies **after** the *make* command has already read the makefile.

4.1.12. *User-defined commands*

Most of the tasks described above are handled by the *make* command. More complex programming tasks can also be defined by adding extra instructions to each makefile. However, rather than modify every program and library makefile in a project individually, the user can get *mkmf* to use alternative 'p.Makefile' and 'l.Makefile' makefile templates when creating program and library makefiles respectively, if these templates exist in the project 'lib' directory (see fig. 3). The templates for project 'vs' include directives for type checking and version control (see appendix B). It is a worthwhile exercise to compare them against the standard templates shown in appendix A.

4.2. Layered Construction of Software Packages

A complex software package may be built and installed in *layers*[2] as shown in table 3.

Table 3. Layers of software

Layer	Components
1	Shared source code (header files)
2	Shared object code (program libraries)
3	Programs
4	Data files
5	Documentation (reference manual, user's guide, etc.)

Each layer is assigned a specific label so that it can be built individually, as well as a priority level so that the complete software package can be constructed in a predetermined sequence. Layers are implemented by attaching type labels to the directories which are part of the building process. Table 4 suggests a set of type labels for each layer. By convention, type label 'update' is used for the construction of the entire software package.

If the project 'vs' is organized into layers in the manner shown in figure 11, then the command

```
% pexec -Tlibsrc make update
```

brings the program libraries up-to-date, while the command

```
% pexec -Tcmdsrc make update
```

Table 4. Layer type labels and priority levels

Layer	Layer Label	'update' Priority Range	
		Lower Limit	Upper Limit
1	include	update.100	update.199
2	libsrc	update.200	update.299
3	cmdsrc	update.300	update.399
4	data	update.400	update.499
5	doc	update.500	update.599

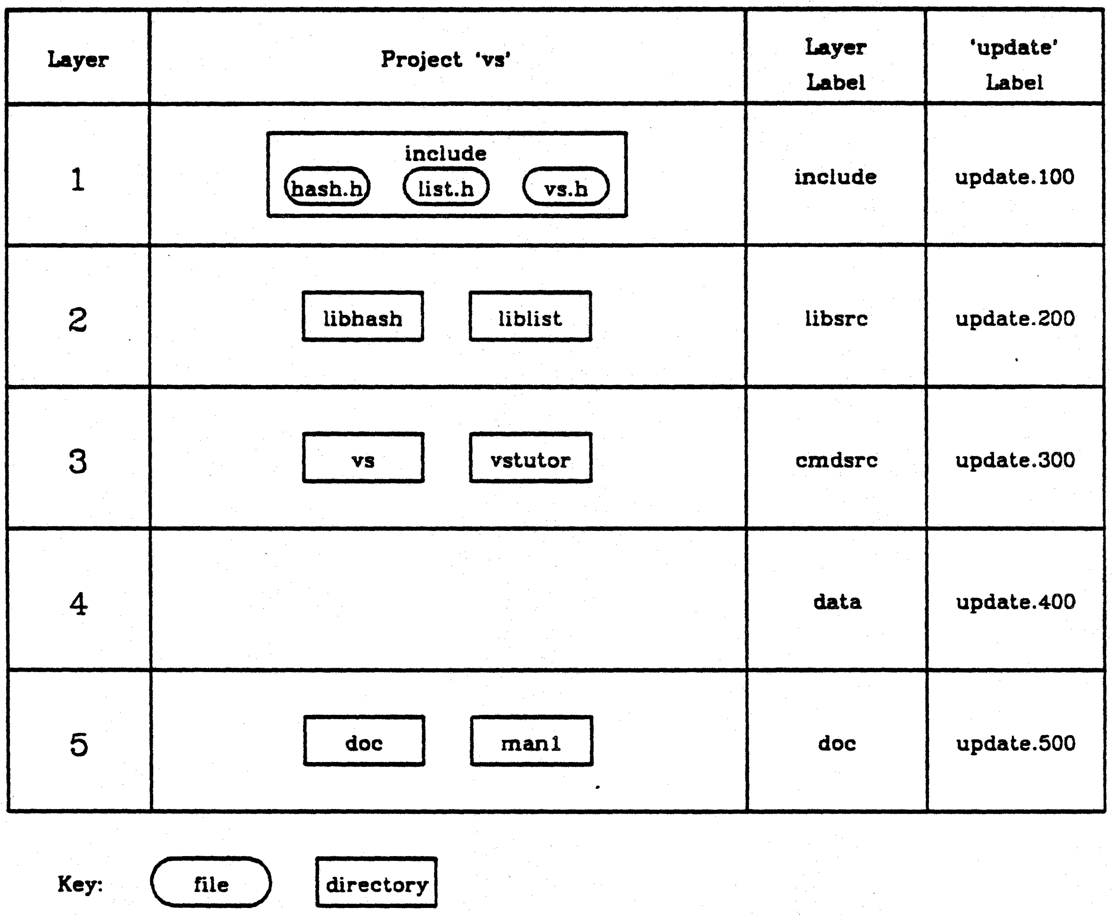


Figure 11. Layers of project 'vs'

does the same for the programs *vs* and *vstutor*. By typing

```
% pexec -T update make update
```

the entire software package can be updated.

4.3. Maintenance of Software Packages

Having established the conventions for maintaining the individual components of a software package, global tasks such as printing, testing, cleaning, etc., can now be described in more detail. The type labels that provide the means for coordinating these tasks are summarized in table 5. Some of the labels have the letter *n* to indicate priority because the directories to which they are attached must be processed in a particular order (see § 2.10.2).

Table 5. Type label conventions

Type Label	Purpose
clean	cleaning up
cmdsrc	program source code
doc	documentation
include	header files
lib	library object code, data
libsrc	library source code
man.*n*	reference manual
print.*n*	printing
project	project root directory
src	source code
test.*n*	testing
update.*n*	installing or updating

4.3.1. *Counting of source lines*

The total number of lines of source code in a software package can be counted by concatenating the source files in each 'src' directory and piping them to the word count program as

```
% pexec -q -Tsrc make PRINT=cat print | wc -l
```

where **-q** suppresses the printing of project directory titles, and **-l** tells *wc* to count lines only.

4.3.2. *Cataloging of functions*

A list of functions, together with the line number and file in which each is defined, may be obtained[13] for each program in a software package by the command

```
% pexec -Tcmdsrc make index
```

A comprehensive index of all the library functions can be generated by outputting the function definitions in each library (e.g. 'libhash' and 'liblist') to the *sort* program by

```
% pexec -q -Tlibsrc make index | sort
```

4.3.3. *Printing*

After printing all of the source code in a project by

```
% pexec -Tprint make print | lpr
```

a table of contents can be produced by

```
% pexec -Tprint make \"PRINT=ls -C\" print | lpr
```

Backslash '\' characters prevent the double quotes '"' from being stripped from the PRINT macro definition by the shell before the *make* command is executed in each directory.

[13] For C, Fortran, and Pascal programs only.

4.3.4. *Program checking*

Software packages written in C can be cross-checked for function and type usage by applying the *lint* command to the source code in each of the project directories containing a program or library

```
% pexec "-Tlibsrc|cmdsrc" make lint
```

4.3.5. *Testing*

All of the tests previously prepared for a software package are exercised by the command

```
% pexec -Ttest ptest > testlog
```

in the directories labeled 'test'. In the case of project 'vs', the following directories are labeled 'test'

Directory	'test' Type Label
^vs^work	test.2
^vs^libhash^work	test.1
^vs^liblist^work	test.1

The outcome of each test – i.e. whether it passes or fails – is recorded in the file called 'testlog' in the current working directory. If a test fails, the cause of the failure is recorded in a file bearing the name 'E*test*' where *test* is the name of the test, located in the directory where the test is executed.

4.3.6. *Comparing versions*

The method for comparing the source code in two different versions of a project depends on the way in which the versions are stored. If they are stored in separate projects the *pdiff* command can be used to compare the contents of the directories belonging to each of the two projects. For example, if 'nvs' is a new version of the project 'vs', then the command

```
% pdiff -Tsrc ^vs ^nvs
```

will produce a summary of the differences between them. However, if the different versions are stored as deltas in a version control system such as *SCCS* or *RCS,* then either the *sccsdiff* command or the *rcsdiff* command must be used instead. To show how this is done with *RCS,*

```
% pexec -Tsrc make VERSION=V2 diff
```

compares the current working version of the source code in the project 'vs' with a previous version labeled 'V2'.

4.3.7. *Releases*

If the source code for a software package is stored in a version control system like *RCS,* it is possible to re-create any particular version of the package provided that all the source files in that version have previously been stamped with a symbolic version name[14] (for example, 'V2'). The process is carried out in two stages. After removing the current version of the source code (hopefully, it has already been stored in the version control system) by the following command sequence,

```
% pexec -Tsrc 'rm `make PRINT=echo print` Makefile'
```

the makefile and source files for the desired version (say 'V2') are checked out by

```
% pexec -Tsrc 'co -rV2 Makefile; make VERSION=V2 co'
```

[14] *SCCS* does not have this feature.

4.3.8. *Cleaning up*

If a software package is in a stable state – that is, it is not being modified – then, as an economy measure, the amount of space that it takes up can be reduced by removing object files plus any other files that can be regenerated easily. This task is implemented by

```
% pexec -Tclean make clean
```

assuming that the directories containing the files to be removed are labeled 'clean'.

5. Retrofitting of Software Projects

Since SPMS accepts an arbitrary directory arrangement, existing software packages can be converted into projects with minimal reorganization. For example, the Fortran 77 compiler project shown in figure 12 was brought under project control by the following commands.

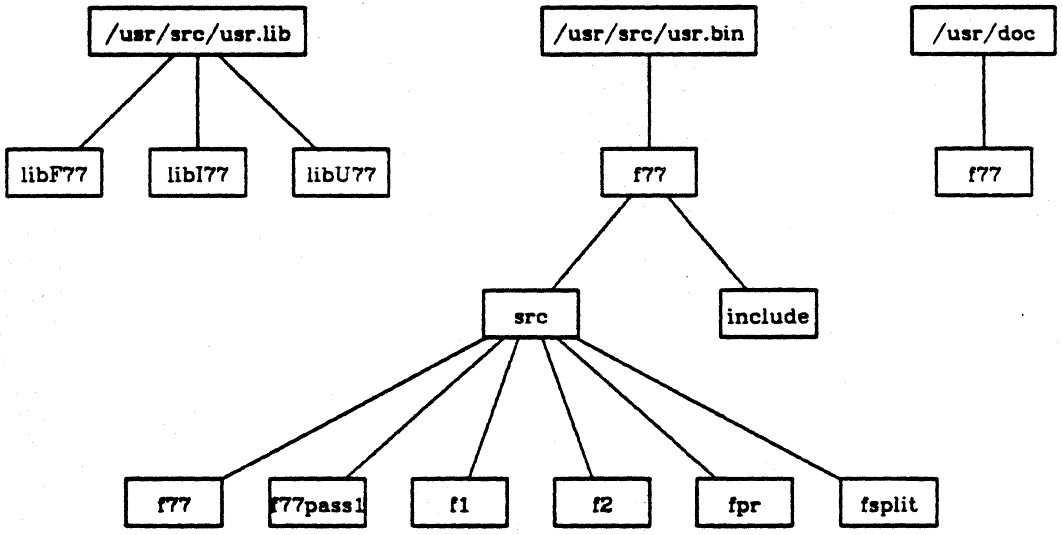


Figure 12. The Fortran 77 compiler project

```
% cd /usr/src/usr.bin
% mkproject -Tproject f77
f77: description? (1 line): Fortran 77
% chproject f77
% pmkdir -Tinclude,src include
include: description? (1 line): header files
% pmkdir src
src: description? (1 line): source code
% pd src
% pmkdir -Tcmdsrc,src,update.300,clean f77 fpr fsplit
f77: description? (1 line): f77 - Fortran 77 compiler
fpr: description? (1 line): fpr - print Fortran files
fsplit: description? (1 line): fsplit - split multi-routine Fortran files
% pmkdir -Tcmdsrc,src,update.300,clean f77pass1 f1 f2
f77pass1: description? (1 line): f77 parser
f1: description? (1 line): f77 code generator
f2: description? (1 line): f77 peephole optimizer
```

```
% cd /usr/src/usr.lib
% pmkdir -Tlibsrc,src,update.200,clean libF77 libI77 libU77
libF77: description? (1 line): f77 function library
libI77: description? (1 line): f77 I/O library
libU77: description? (1 line): f77 system utility library
% pmkdir -Ndoc -Tdoc /usr/doc/f77
/usr/doc/f77: description? (1 line): f77 documentation
```

The directory aliasing feature of SPMS is also demonstrated by this example. Sometimes a project will have more than one directory with the same name as is the case with the Fortran project, where the name of the 'src/f77' compiler driver program directory coincides with the name of the 'doc/f77' documentation directory. Since SPMS insists that the directories within each project have unique names, the 'doc/f77' directory is aliased to 'doc' using *pmkdir -N*.

6. Acknowledgements

SPMS was originally developed in response to a need for managing a suite of programs for modeling the dynamic behavior of the piping in nuclear power stations in the event of sudden rupture. The project, involving 10 programmers, over 2000 files, and upwards of 100,000 lines of Fortran code, was directed by Graham H. Powell, Professor of Civil Engineering at U.C. Berkeley. His encouragement, support, and criticism has been invaluable.

The Computer Systems Research Group, directed by Professor R. S. Fabry, provided the resources for the development of the second and third versions of SPMS. David Mosher offered valuable suggestions about the system and this report. Stuart Feldman also provided helpful comments.

John Foderero exercised SPMS on the Franz Lisp System and Jim Kleckner did the same for VLSI CAD software. Mike O'Dell installed the system in Italy. Their help in debugging SPMS is much appreciated.

Edward Wang modified *lint* so that it could generate lint libraries. His work enabled SPMS to check complete software packages for consistency.

7. References

[1] Arnold, K., "Screen Updating and Cursor Movement Optimization: A Library Package", Computer Science Division, EECS, University of California, Berkeley.

[2] Cristofor, E., Wendt, T. A., and Wonsiewicz, B. C., "Source Control + Tools = Stable Systems", *Proceedings of the Fourth Computer Software and Applications Conference,* pp. 527-532, October 29-31, 1980.

[3] Feldman, S. I., "Make - A Program for Maintaining Computer Programs", *Software - Practice and Experience,* vol. 9, no. 4, pp. 255-265, April 1979.

[4] Johnson, S. C., "Lint, a C Program Checker", *The UNIX Programmer's Manual,* Bell Laboratories, July 1978.

[5] Kernighan, B. W., and Mashey, J. R., "The Unix Programming Environment", *Computer,* vol. 14, no. 4, April 1981.

[6] Rochkind, M. J., "The Source Code Control System"", *IEEE Transactions on Software Engineering,* vol. SE-1, no. 4, pp. 364-370, December 1975.

[7] Shafer, S., Accetta, M., Gosling J., Lucas, B., and Zsarnay, J., "Managing UNIX: Obtaining a powerful portable programming environment under UNIX", Computer Science Dept., Carnegie-Mellon University, December 1979.

[8] Shafer, S., "Maintaining UNIX Projects: Creating and Updating Shared Software under UNIX", Computer Science Dept., Carnegie-Mellon University, June 1980.

[9] Shoens, K., "The Mail Reference Manual", *The UNIX Programmer's Manual,* vol. 2c, Computer Science Division, EECS, University of California, Berkeley, September 1982.

[10] Tichy, W.F., "Design, Implementation, and Evaluation of a Revision Control System", *Proceedings of the Sixth International Conference on Software Engineering.* pp. 58-67, September 1982.

Appendix A. Standard Makefile Templates

```
DEST        = .

EXTHDRS     =

HDRS        =

LDFLAGS     =

LIBS        =

LINKER      = cc

MAKEFILE    = Makefile

OBJS        =

PRINT       = pr

PROGRAM     = a.out

SRCS        =

all:            $(PROGRAM)

$(PROGRAM):     $(OBJS) $(LIBS)
                @echo -n "Loading $(PROGRAM) ... "
                @$(LINKER) $(LDFLAGS) $(OBJS) $(LIBS) -o $(PROGRAM)
                @echo "done"

clean:;         @rm -f $(OBJS)

depend:;        @mkmf -f $(MAKEFILE) PROGRAM=$(PROGRAM) DEST=$(DEST)

index:;         @ctags -wx $(HDRS) $(SRCS)

install:        $(PROGRAM)
                @echo Installing $(PROGRAM) in $(DEST)
                @install -s $(PROGRAM) $(DEST)

print:;         @$(PRINT) $(HDRS) $(SRCS)

program:        $(PROGRAM)

tags:           $(HDRS) $(SRCS); @ctags $(HDRS) $(SRCS)

update:         $(DEST)/$(PROGRAM)

$(DEST)/$(PROGRAM): $(SRCS) $(LIBS) $(HDRS) $(EXTHDRS)
                @make -f $(MAKEFILE) DEST=$(DEST) install
```

Figure A1. 'p.Makefile' program makefile template

```
DEST          = .

EXTHDRS       =

HDRS          =

LIBRARY       = lib.a

MAKEFILE      = Makefile

OBJS          =

PRINT         = pr

SRCS          =

all:          $(LIBRARY)

$(LIBRARY):   $(OBJS)
              @echo -n "Loading $(LIBRARY) ... "
              @ar cru $(LIBRARY) $(OBJS)
              @ranlib $(LIBRARY)
              @echo "done"

clean:;       @rm -f $(OBJS)

depend:;      @mkmf -f $(MAKEFILE) LIBRARY=$(LIBRARY) DEST=$(DEST)

extract:;     @ar xo $(DEST)/$(LIBRARY)
              @rm -f __.SYMDEF

index:;       @ctags -wx $(HDRS) $(SRCS)

install:      $(LIBRARY)
              @echo Installing $(LIBRARY) in $(DEST)
              @install $(LIBRARY) $(DEST)
              @ranlib $(DEST)/$(LIBRARY)

library:      $(LIBRARY)

print:;       @$(PRINT) $(HDRS) $(SRCS)

tags:         $(HDRS) $(SRCS); @ctags $(HDRS) $(SRCS)

update:       $(DEST)/$(LIBRARY)

$(DEST)/$(LIBRARY): $(SRCS) $(HDRS) $(EXTHDRS)
              @-ar xo $(DEST)/$(LIBRARY)
              @make -f $(MAKEFILE) DEST=$(DEST) install clean
```

Figure A2. 'l.Makefile' library makefile template

Appendix B. Project 'vs' Makefile Templates

```
CFLAGS      = -D$(VERSION) -I../../include -O

DEST        = ../../bin

EXTHDRS     =

HDRS        =

LDFLAGS     =

LIBS        = ../../lib/libhash.a \
              ../../lib/liblist.a \
              /usr/lib/libcurses.a \
              /usr/lib/libtermlib.a

LINKER      = cc

LINTFLAGS   = -D$(VERSION) -I../../include

LINTLIST    = ../../lib/llib-lhash.ln \
              ../../lib/llib-llist.ln \
              -lc

MAKEFILE    = Makefile

OBJS        =

PRINT       = pr

PROGRAM     =

SRCS        =

VERSION     = V3

all:          $(PROGRAM)

$(PROGRAM):   $(OBJS) $(LIBS)
              @echo -n "Loading $(PROGRAM) ... "
              @$(LINKER) $(LDFLAGS) $(OBJS) $(LIBS) -o $(PROGRAM)
              @echo "done"

clean:;       @rm -f $(OBJS)

co:;          @co -r$(VERSION) $(HDRS) $(SRCS)

depend:;      @mkmf -f $(MAKEFILE) PROGRAM=$(PROGRAM) DEST=$(DEST)

diff:;        @rcsdiff -r$(VERSION) $(HDRS) $(SRCS)

index:;       @ctags -wx $(HDRS) $(SRCS)

install:      $(PROGRAM)
              @echo Installing $(PROGRAM) in $(DEST)
              @install -s $(PROGRAM) $(DEST)

lint:;        @lint $(LINTFLAGS) $(SRCS) $(LINTLIST)
```

```
print:;             @$(PRINT) $(HDRS) $(SRCS)

program:            $(PROGRAM)

tags:               $(HDRS) $(SRCS); @ctags $(HDRS) $(SRCS)

update:             $(DEST)/$(PROGRAM)

$(DEST)/$(PROGRAM): $(SRCS) $(LIBS) $(HDRS) $(EXTHDRS)
                    @make -f $(MAKEFILE) DEST=$(DEST) install tags
```

Figure B1. 'p.Makefile' program makefile template

```
CFLAGS      = -D$(VERSION) -I../../../include -O

DEST        = ../../../lib

EXTHDRS     =

HDRS        =

LIBNAME     =

LIBRARY     = lib$(LIBNAME).a

LINTFLAGS   = -D$(VERSION) -I../../../include

LINTLIB     = llib-l$(LIBNAME).ln

LINTLIST    = -lc

MAKEFILE    = Makefile

OBJS        =

PRINT       = pr

SRCS        =

VERSION     = V3

all:          $(LIBRARY)

$(LIBRARY):   $(OBJS) $(LINTLIB)
              @echo -n "Loading $(LIBRARY) ... "
              @ar cru $(LIBRARY) $(OBJS)
              @ranlib $(LIBRARY)
              @echo "done"

$(LINTLIB):   $(SRCS) $(HDRS) $(EXTHDRS)
              @echo "Loading $(LINTLIB) ..."
              @lint $(LINTFLAGS) -C$(LIBNAME) $(SRCS)
              @echo "done"

clean:;       @rm -f $(OBJS)

co:;          @co -r$(VERSION) $(HDRS) $(SRCS)

depend:;      @mkmf -f $(MAKEFILE) LIBRARY=$(LIBRARY) DEST=$(DEST)

diff:;        @rcsdiff -r$(VERSION) $(HDRS) $(SRCS)

extract:;     @ar xo $(DEST)/$(LIBRARY)
              @rm -f __.SYMDEF

index:;       @ctags -wx $(HDRS) $(SRCS)

install:      $(LIBRARY)
              @echo Installing $(LIBRARY) in $(DEST)
              @install $(LIBRARY) $(DEST)
              @ranlib $(DEST)/$(LIBRARY)
              @echo Installing $(LINTLIB) in $(DEST)
```

```
                @install $(LINTLIB) $(DEST)

library:        $(LIBRARY)

lint::          @lint $(LINTFLAGS) $(SRCS) $(LINTLIST)

lintlib:        $(LINTLIB)

print::         @$(PRINT) $(HDRS) $(SRCS)

tags:           $(HDRS) $(SRCS); @ctags $(HDRS) $(SRCS)

update:         $(DEST)/$(LIBRARY)

$(DEST)/$(LIBRARY): $(SRCS) $(HDRS) $(EXTHDRS)
                @-ar xo $(DEST)/$(LIBRARY)
                @make -f $(MAKEFILE) DEST=$(DEST) install tags clean
```

Figure B2. 'l.Makefile' library makefile template

Appendix C. Project Pathname Syntax

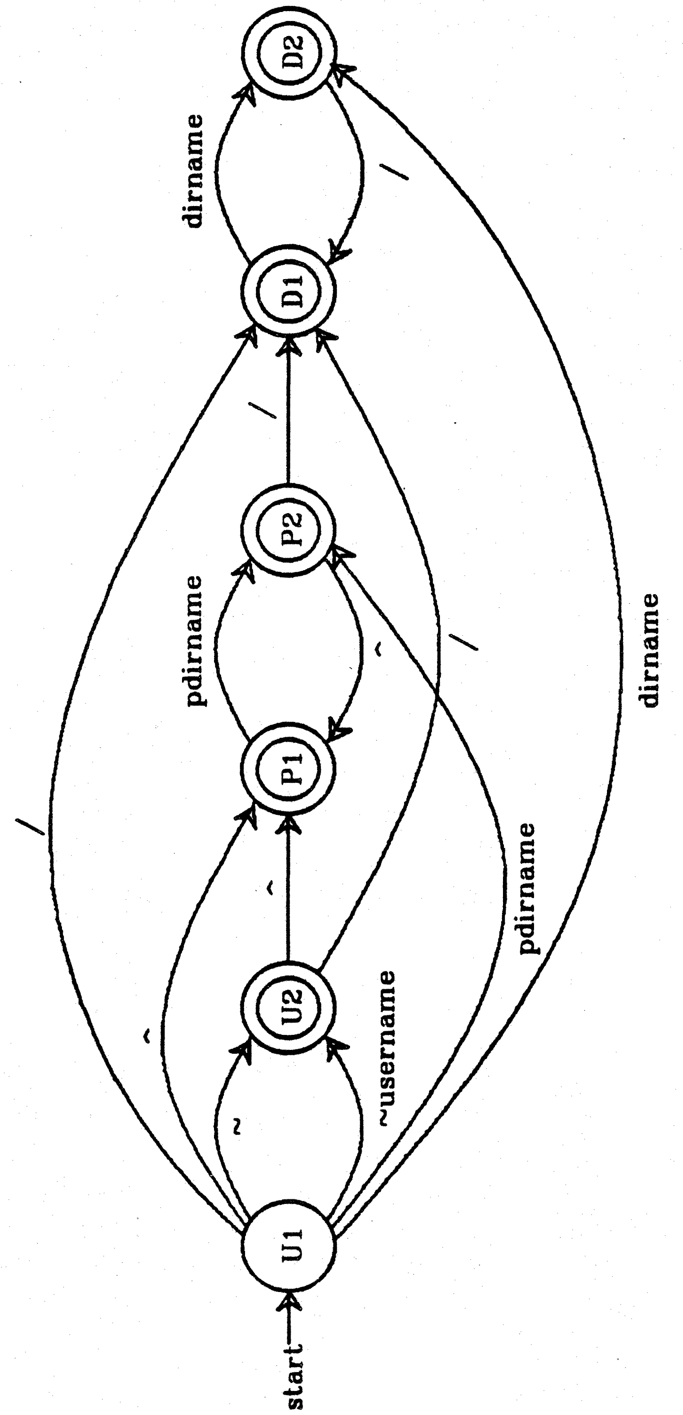


Figure C1. Project pathname syntax

Appendix D. SPMS Command Summary

chproject	activate project environment
mkmf	makefile editor
mkproject	make a project root directory
pcp	copy files
pd	change working project directory
pdiff	differential project comparator
pexec	execute command over project hierarchy
pfind	find files in projects
pgrep	search files in a project hierarchy for a pattern
phelp	on-line help
plog	records progress of a project
pman	print project manual
pmkdir	make a project directory
pmv	move or rename files
ppd	list project directories
prmdir	remove a project directory
ptest	test a project module
pwp	print working project name
rmproject	remove a project root directory

NAME

spmsintro - introduction to SPMS commands

INTRODUCTION

The Software Project Management System (SPMS) is a system for the management of medium- to large-scale software systems. SPMS provides, within the UNIX environment, a number of commands which can greatly simplify many tasks associated with program development and maintenance. SPMS does not attempt to duplicate existing UNIX program development tools such as *make* or *SCCS,* but instead provides a way of coordinating these tools.

Each software package managed by SPMS is organized as a project. A project is a collection of directories, each of which supports a specific activity such as program development, testing, or documentation. There is no restriction on the number of directories belonging to a project. The directory layout is arbitrary, and can be altered to reflect the changing needs of the package. Projects can be nested to any level and a mechanism is provided for executing commands globally over an entire project hierarchy.

DESCRIPTION

Getting Started

Before using SPMS for the first time you must do the following –

If you are a **C shell**, (*csh*), user:

1. Include the directory '/usr/new' in the command search path. This is done by altering the PATH variable in one of the startup files, '.cshrc' or '.login', in the home directory.
2. Add the following aliases to the '.cshrc' file
   ```
   alias chproject 'eval `"chproject" \!*`'
   alias pd 'eval `"pd" \!*`'
   ```
3. Add the following command to the '.login' file
   ```
   chproject ^
   ```
4. Convert the home directory to a project root directory by typing
   ```
   /usr/new/mkproject -d ^
   ```
5. Execute the '.cshrc' and '.login' files by typing
   ```
   source .cshrc
   source .login
   ```

If you are a **Bourne shell,** (*sh*), user:

1. Include the directory '/usr/new' in the command search path. This is done by altering the PATH variable in the startup file, '.profile', in the home directory.
2. Add the following command to the '.profile' file:
   ```
   eval `chproject \^`
   ```
3. Convert the home directory to a project root directory by typing
   ```
   /usr/new/mkproject -d \^
   ```
4. Execute the '.profile' file by typing
   ```
   . .profile
   ```

Global Operations

The means for executing a command over an entire software package is provided by the *pexec* command. By labeling each project directory according to the type of activity that it supports, global operations can be restricted to specific directories. These labels are known as *type labels.* In some instances, the directories affected by a global command must be processed in a particular order. This ordering is achieved by appending priorities to type labels. For example, if the project directories 'include', 'cmd1', 'cmd2', 'lib1', and 'lib2' have the following labels,

include	print.0, src, update.100, include
cmd1	print.1, src, update.300, cmdsrc
cmd2	print.1, src, update.300, cmdsrc
lib1	print.2, src, update.200, libsrc
lib2	print.2, src, update.200, libsrc

the entire software package can be updated by the command

```
pexec –Tupdate  make update
```

in the order 'include', 'lib1', 'lib2', 'cmd1', 'cmd2'.

Global commands can be made even more precise by using boolean expressions on type labels to select project directories. The formal definition of a boolean type label expression is

$$E \rightarrow E \textbf{ or } E \mid E \textbf{ and } E \mid \textbf{not } E \mid (\mathrm{E}) \mid \textbf{id}$$

where E is a boolean expression; **and**, **or**, and **not** are boolean operators; and **id** is a type label. **Or** and **and** are left-associative. **Or** has the lowest precedence, then **and**, then **not**. In terms of entering the boolean expression on the command line, '**or**' is represented by the character '|', '**and**' by the character '&', and '**not**' by '!'. Since these characters, together with '(' and ')', are meaningful to the command shell, it is good idea to enclose the whole expression in quotes. However, even if this is done, the '!' character must still be escaped by a backslash '\' if it precedes a type label to prevent it from being interpreted by the *csh* history mechanism. The command

```
pexec "–Tprint&(libsrc | cmdsrc)" 'pr *.h *.c' | lpr
```

prints the source code in the command and library directories, but not the directory containing header files. Alternatively,

```
pexec "–Tprint&\!include" 'pr *.h *.c' | lpr
```

achieves the same result.

Project Pathnames

Project pathnames provide a convenient way for accessing a particular directory or file within a project hierarchy. A project pathname is formed by a succession of project names separated by '^' characters, followed by the name of the directory or file. For instance, to describe a file 'main.c' in the project directory 'work' in the project 'spms', the project pathname is

```
^spms^work/main.c
```

The project at the top of each user's project hierarchy is called the *root project* and is given the special name '^'. If a project pathname begins with the character '^', it is interpreted relative to the root project and is called an *absolute project pathname*. However, a project pathname not beginning with '^' is interpreted with respect to the current working project and is therefore called a *relative project pathname*. The parent of the working project is called '....' and the alternative name for the current project is '...'.

Project pathnames may have a prepended ~*username*, and an appended regular pathname. For example, the pathname

```
~pjn^spms^work/old/main.c
```

represents the path to 'main.c' located in the directory 'old' in the project 'spms' owned by 'pjn'.

OPTIONS

The options to SPMS commands follow certain conventions. Keyword options are uppercase (with the exception of the –f option). The keyword can immediately follow the option, or be separated by an arbitrary amount of space. The following options are uniformly recognized.

-P *pdirname*
  Specify a project other than the current working project.

-T *typexpr*
  Only use project directories corresponding to boolean type label expression, *typexpr*.

Non-keyword options are lowercase (with the exception of the **-D** option). The following options are standard.

-q Quiet mode. Do not print titles.

-r Apply the command recursively to subprojects.

-x Trace and print, but do not execute.

-D Print the expanded pathname when a project pathname is converted to a regular pathname.

An option specified as {+-}*x* means +*x* or -*x*.

ENVIRONMENT VARIABLES

PROJECT	Absolute pathname of the current working project root directory. This variable is set by *chproject*.
ROOTPROJECT	Absolute pathname of the root project directory. The default is the user's home directory.

FILES

...	Project link directory.

SEE ALSO

mkmf(1), chproject(1P), mkproject(1P), pcp(1P), pd(1P), pdiff(1P), pexec(1P), pfind(1P), pgrep(1P), phelp(1P), plog(1P), pman(1P), pmkdir(1P), pmv(1P), ppd(1P), prmdir(1P), ptest(1P), pwp(1P), rmproject(1P)

Peter J. Nicklin *The SPMS Software Project Management System*

AUTHOR

Peter J. Nicklin

BUGS

At present, project pathnames are only recognized by SPMS commands.

Since the Bourne shell, *sh,* recognizes the '^' character as an alternative pipe symbol, Bourne shell users must type '\^' instead.

NAME
chproject – activate project environment

SYNOPSIS
chproject [–df] projectname

DESCRIPTION
Chproject activates the project environment for *projectname.* The project then becomes the current working project.

After activating the environment, *chproject* changes to the root directory of *projectname* and executes commands from the '.projectrc' file in that directory.

OPTIONS
–d	Print the project description.
–f	Instruct *chproject* to ignore the '.projectrc' file.

ENVIRONMENT VARIABLES
PROJECT	Current working project root directory.
SHELL	Name of command interpreter.

FILES
.projectrc	Command file executed by *chproject.*

SEE ALSO
csh(1), mkproject(1P), sh(1)

DIAGNOSTICS
Exit status 0 is normal. Exit status 1 indicates an error.

AUTHOR
Peter J. Nicklin

BUGS
C shell, *csh,* users should be aware that *chproject* is an aliased command. The '.cshrc' file in your home directory should contain the following alias:

```
alias chproject 'eval `"chproject" \!*`'
```

Bourne shell, *sh,* users must give the *chproject* command as:

```
eval `chproject [–df] projectname`
```

NAME

mkproject – make a project root directory

SYNOPSIS

mkproject [{+–}**d**] [{+–}N alias] [{+–}T type[,type ...]] projectname ...

DESCRIPTION

Mkproject creates a directory called *projectname.* The directory is known as a *project root directory* and is the focus for a project. Standard entries, '.', for the directory itself, '..', for its parent, and '...' for the project link directory, are made automatically. After the directory has been created, *mkproject* prompts the user for a line describing the purpose of the project.

If the name of the directory conflicts with an existing project, an alternative alias for the project may be specified via the –N option. However, even if this option is used, the name of the directory will be recognized as a project unless it is disguised as a regular pathname. For example, to create another project called 'spms' with alias 'newspms', type:

```
mkproject  –N newspms  ./spms
```

Mkproject may also be used to convert an existing regular directory to a project root directory.

Mkproject requires write permission in the parent directory.

OPTIONS

+d Change the description of an existing project.

–d Turn off prompting for the description of a new project.

N *alias*
Change the alias of the project.

–N *alias*
Specify an alternative alias for a new project.

T *type* Add a type label to an existing project root directory. If the type label already exists but has a different priority, then it must be removed using the *rmproject* command.

–T *type*
Specify a type label for a new project root directory.

FILES

...	Project link directory.
..._temp	Temporary project link directory.

LIMITATIONS

Project descriptions can be no longer than 128 characters.

SEE ALSO

mkdir(1), rmproject(1P)

DIAGNOSTICS

The error message, "mkproject: *project*/... temporarily unavailable", indicates that a '..._temp' temporary project link directory exists. This could be because another user is altering the project link directory, or because a system crash terminated *mkproject* prematurely. If the latter case, then removing the temporary file will fix the problem.

Exit status 0 is normal. Exit status 1 indicates an error.

AUTHOR

Peter J. Nicklin

BUGS

The root project, '^', cannot have an alternative alias.

Directory aliases must not include the characters ':' or '|'.

Type labels must not include the characters ':' or '/'.

NAME

pcp – copy files

SYNOPSIS

pcp [**–i**] file1 file2

pcp [**–i**] file ... dirname

DESCRIPTION

Pcp copies *file1* onto *file2*. The mode and owner of *file2* are preserved if it already exists, otherwise the mode of the source file is used.

In the second form, one or more *files* are copied into *dirname* with their original file names.

File and *dirname* may be either regular or project pathnames. However, because *pcp* interprets both *file* and *dirname* arguments as project pathnames, if *file* matchs the name of a project directory within the same project, then *pcp* will print the error message 'pcp: can't copy project directory *file*', unless *file* is disguised as *./file.*

Pcp blindly overwrites existing files unless the **–i** option is specified.

Pcp refuses to copy a file onto itself.

OPTIONS

–i Interactive mode. *Pcp* will prompt the user with the name of the file whenever the copy will cause an old file to be overwritten. An answer of 'y' will cause *pcp* to continue. Any other answer will prevent it from overwriting the file

SEE ALSO

cp(1)

DIAGNOSTICS

Exit status 0 is normal. Exit status 1 indicates an error.

AUTHOR

Peter J. Nicklin

NAME

pd – change working project directory

SYNOPSIS

pd [**–dp**] [dirname]

DESCRIPTION

Dirname becomes the new working directory. *Dirname* may be either a project or a regular directory.

Given without any arguments, *pd* returns you to the root directory of the current working project.

If *dirname* is a project directory in another project, *pd* makes that project the current working project.

OPTIONS

–d Print project directory description.

–p Push old working directory onto the directory stack. The current working project is not changed.

EXAMPLE

To change to the 'work' directory of a project named 'spms':

```
pd ^spms^work
```

SEE ALSO

cd(1), csh(1)

DIAGNOSTICS

Exit status 0 is normal. Exit status 1 indicates an error.

AUTHOR

Peter J. Nicklin

BUGS

Pd is provided only for C shell, *csh,* users because it is an aliased command. The '.cshrc' file in your home directory should contain the following alias:

```
alias pd 'eval `"pd" \!*`'
```

NAME

pdiff – differential project comparator

SYNOPSIS

pdiff [–rx] [–T typexpr] [diff options] projectname1 projectname2

pdiff [–x] [diff options] pdirname1 pdirname2

pdiff [–x] [diff options] file1 file2

DESCRIPTION

Pdiff compares files in projects using the *diff* command. *Diff* tells what lines must be changed in the files to bring them into agreement. Except in rare circumstances, it finds a smallest sufficient set of file differences.

If both arguments are projects, *pdiff* sorts the project directories in each project by name, and then runs *diff* on the contents of common directories. Binary files that differ, common subdirectories, and files that appear in only one directory are listed.

If *pdirname1* is a project, then a project directory in that project with the same name as *pdirname2* is used (and vice versa).

If *file1* is a project, then a file in that project with the same name as *file2,* residing in a project directory with the same name as the current working project directory, is used (and vice versa).

OPTIONS

–r Apply *pdiff* recursively to common subprojects.

–x Trace and print *diff* commands, but do not execute.

–T *typexpr*
Only compare project directories corresponding to boolean type label expression, *typexpr.*

SEE ALSO

diff(1)

DIAGNOSTICS

The error message "pdiff: don't know which project directory to use in *projectname*" indicates that the file or directory that is being compared against *projectname* is not part of the current working project.

Exit status is 0 for no differences, 1 for some, 2 for trouble.

AUTHOR

Peter J. Nicklin

NAME

pexec – execute command over project hierarchy

SYNOPSIS

pexec [–?ciqx] [–P pdirname] [**–T** typexpr] [**–X** errstatus] command

DESCRIPTION

Pexec descends recursively through a project hierarchy executing *command* in each project directory using either the *csh* or *sh* command interpreter. The directories at each level are traversed in alphabetical order.

Before executing *command* in each directory, the current working project is reset to the project in which the directory resides.

Unless the **–i** option is used, *pexec* quits if a directory is inaccessible or *command* returns a non-zero exit status. This prevents propagation of errors through a project.

Care should be taken when using the characters $ * [| ( ) and \ in the *command* as they are also meaningful to the command shell. It is safest to enclose the entire *command* in single quotes.

If a *typexpr* boolean type label expression is specified, *pexec* considers only those project directories with type labels that satisfy that expression. The order in which the project directories are traversed depends on the relative priorities of the type labels attached to each directory. Only those type labels that appear in *typexpr* are used. Directories with labels of the same priority are sorted alphabetically. For example, if the project directories 'include', 'cmd1', 'cmd2', 'lib1', and 'lib2' have the following labels:

include	print.0, src, update.0, include
cmd1	print.1, src, update.2, cmdsrc
cmd2	print.1, src, update.2, cmdsrc
lib1	print.2, src, update.1, libsrc
lib2	print.2, src, update.1, libsrc

then type label expression 'update' will force *pexec* to traverse the directories in the order 'include', 'lib1', 'lib2', 'cmd1', 'cmd2'.

Labels that are part of a negated expression are not used for sorting.

OPTIONS

–? Do not print "Do you really want to quit? [yn](y):" when interrupted in foreground mode. Quit immediately.

–c Instruct *csh* to read the '.cshrc' startup file.

–i Ignore inaccessible directories and non-zero exit codes from *command.*

–q Quiet mode. Do not print '==> *directory* <==' titles.

–x Trace, and print directory titles, but do not execute *command.*

–P *pdirname*
Specify a project other than the current working project. If *pdirname* is a project directory, *command* will be executed only in that directory.

–T *typexpr*
Only execute *command* in project directories corresponding to boolean type label expression, *typexpr.*

–X *errstatus*
If *pexec* fails, exit with status *errstatus.* Default error status is 1.

EXAMPLES

To list all of the files in a project using *ls*, type:

```
pexec ls
```

If the directories containing source code have been labeled previously as type 'src', then, to count the total number of lines of source code in a project, type:

```
pexec –Tsrc ´cat *.h *.c´ | wc –l
```

where quotes surround the *cat* command to prevent file name expansion in the current directory.

ENVIRONMENT VARIABLES

PROJECT	Current working project root directory.
SHELL	Name of command interpreter.

SEE ALSO

csh(1), sh(1)

DIAGNOSTICS

If *pexec* is interrupted while executing *command* in foreground, the message, "Do you really want to quit? [yn](y):" will appear after *command* has completed.

If the error message, "pexec: *label, label* ...: conflicting type label priorities", occurs when performing an operation on a set of project directories that have been selected according to a boolean type label expression with more than one type label, this indicates that the directories cannot be sorted satisfactorily because of a clash in priorities. For example, if project directories *a* and *b*, selected by type label expression 'print&update', have the following type labels:

directory a:	print.1, src, update.2
directory b:	print.2, src, update.1

the ordering will be *ab* if the directories are sorted according to the 'print' type label, and *ba* if they are sorted by the 'update' type label. The **–D** debug option can be used to dump the list of project directories that match *typexpr*, together with their type labels.

Pexec returns the exit status of *command*. Exit status 0 is normal. Non-zero exit status indicates an error.

AUTHOR

Peter J. Nicklin

BUGS

The PROJECT environment variable must be defined.

Since *pexec* uses a separate command shell to execute *command* in each directory, the characters $ * [| ( ) and \ will be meaningful to that shell even if *command* is protected by single quotes.

NAME

pfind – find files in projects

SYNOPSIS

pfind [–l] [**–P** pdirname] [**–T** typexpr] file ...

DESCRIPTION

Pfind descends recursively through a project hierarchy seeking *files.*

OPTIONS

–l List in long format, giving the full pathname of *file.*

–P *pdirname*
: Specify a project other than the current working project. If *pdirname* is a project directory, *pfind* will search only that directory.

–T *typexpr*
: Only search project directories corresponding to boolean type label expression, *typexpr.*

EXAMPLES

If the file 'core' exists in the project directory 'work' of the current working project 'spms', the command 'pfind core' will print:

```
...^work/core
```

and the command 'pfind -l core' might print something like:

```
/usr/pjn/spms/work/core
```

SEE ALSO

find(1)

DIAGNOSTICS

Exit status 0 is normal. Exit status 1 indicates an error.

AUTHOR

Peter J. Nicklin

BUGS

Should be able to do pattern matching on file names.

NAME

pgrep – search files in a project hierarchy for a pattern

SYNOPSIS

pgrep [**–eilmnw**] [**–f** makefile] [**–C** command] [**–F** patfile] [**–P** pdirname] [**–T** typexpr] [pattern [file ...]]

DESCRIPTION

Pgrep searchs through the files in a project hierarchy for lines matching *pattern.* Normally, each line found is printed to standard output. Alternatively, a *command* can be executed in each project directory, with arguments that are the names of files containing *pattern.*

The names of files can be specified as arguments, or obtained from the 'HDRS' and 'SRCS' macro definitions in a makefile (**–m** option), or a combination of both. When *pgrep* is told to use a makefile and the **–f** option is not present, the files 'makefile' and 'Makefile' are tried in order.

Pgrep uses *pexec* to execute either the *grep* or *egrep* commands over a project hierarchy. *Grep* patterns are limited to regular expressions in the style of *ex*(1). *Egrep* patterns are full regular expressions. Care should be taken when using the characters $ * [| ( ) and \ in *pattern* as they are also meaningful to the command shell. It is safest to enclose the entire *pattern* in single quotes.

OPTIONS

–e Use *egrep* instead of *grep.*

–f *makefile*
Specify an alternative *makefile* file name. This option also implies the **–m** option.

–i Ignore case of letters when making comparisons (i.e. upper and lower case are considered identical). *Grep* only.

–l List the names of files with matching lines. The file names are printed one per line.

–m Obtain the names of files to search from a makefile. If no **–f** option is present, the makefiles 'makefile' and 'Makefile' are tried in order.

–n Precede each matching line by its relative line number in the file.

–w Treat *pattern* as a word (i.e. as if surrounded by '\<' and '\>'; see *ex*(1)). *Grep* only.

–C *command*
Execute *command* in each project directory, with arguments that are the names of files containing *pattern.*

–F *patfile*
The regular expression is taken from *patfile. Egrep* only.

–P *pdirname*
Specify a project other than the current working project. If *pdirname* is a project directory, search files in that directory only.

–T *typexpr*
Only search files in project directories corresponding to boolean type label expression, *typexpr.*

EXAMPLES

If all the directories in a project that contain source code have been labeled previously as type 'src', then, to search all the source code makefiles for the pattern 'VERSION =', type:

```
pgrep –Tsrc 'VERSION.*=' Makefile
```

where quotes surround the pattern to prevent file name expansion in the current directory.

To edit all the source code files that contain the pattern 'open(' using the *vi* editor, type:

```
pgrep -m -Cvi -Tsrc 'open('
```

where **-m** tells *pgrep* to get the names of the source code files from a makefile.

FILES

makefile	Default makefile.
Makefile	Alternative default makefile.

SEE ALSO

egrep(1), ex(1), grep(1), make(1), mkmf(1P), pexec(1P), pgrep(3P), vi(1)

DIAGNOSTICS

Exit status is 0 if any matches are found, 1 if none, 2 for syntax errors or inaccessible files and directories.

AUTHOR

Peter J. Nicklin

NAME

phelp – on-line help for a project

SYNOPSIS

phelp [**–P** projectname] [topic [subtopic ...]]

DESCRIPTION

Phelp (*pē'help*) provides information about a project. There are two modes of operation:

Interactive

After the *phelp* command is typed, it will respond with '???' indicating that it is ready for a command. The following commands are recognized:

Command	Response
a topic name	Print help information on topic.
index	Display list of topics available at this level.
help	Display help on how to use *phelp.*
?	Display this command summary.
q	Exit from *phelp.*
P *projectname*	Change to another project.
˜	Return to the top level of help topics.
carriage return only	Go up one level of help topics.

If a topic name is typed in reply, *phelp* will print a page of information and then wait until a space is typed before it continues.

Command line topics

Information on a specific *topic* can be requested by giving the *phelp* command with the topic name as an argument on the command line. If the last argument is **index,** the subtopics corresponding to the previous argument are listed. Provided the information has not been redirected to a file, *phelp* will prompt for further commands once it has finished printing.

Creating topics

Project help files reside in the 'help' directory located in the project root directory. Subtopics are contained in subdirectories. A subtopic directory has the same name as a help file but with a '.d' suffix. For example, if "bugfixes" is a subtopic of "news", it will be found in the directory 'news.d', which itself is a subdirectory of the project 'help' directory.

Subtopics can be nested to any level.

Any filename beginning with a '.' is ignored by *phelp.*

OPTIONS

–P *projectname*
: Specify a project other than the current working project.

EXAMPLES

In the following examples, **bold** script indicates what the user types.

To find out about topic "news":

% **phelp**
(prints introduction to phelp and a list of topics available)
??? **news**
(prints "news")
??? **q**
%

Using command line arguments instead:

```
% phelp news
(prints "news")
??? q
%
```

To print topic "news" on the line printer:

```
% phelp news | lpr
%
```

If "bugfixes" is a subtopic of "news", then to print "news", and then "bugfixes":

```
% phelp news
(prints "news")
news subtopics:  bugfixes
news-->??? bugfixes
(prints "bugfixes")
news-->??? q
%
```

FILES

/usr/new/lib/phelp.help	Introduction on how to use *phelp.*
/usr/new/lib/phelp.cmd	*Phelp* command summary.
project/help/*	Help text files.
project/help/*.d	Subtopic directories.

DIAGNOSTICS

Exit status 0 is normal. Exit status 1 indicates an error.

AUTHOR

Peter J. Nicklin

NAME

plog – record progress of a project

SYNOPSIS

plog [**–e**] [{+–}**h**] [**–p**[low[–high]]] [**–s**] [projectname]

DESCRIPTION

Plog is an intelligent electronic notebook system based upon the *Mail* program.

Plog records messages by invoking the *Mail* program. After the *Mail* program starts up, you are expected to type in your message, followed by a period or a CTRL-D at the beginning of a line. If *plog* is invoked with a *projectname* argument the message is appended to the 'projectlog' file in the root directory of that project; otherwise the current working project is assumed.

The 'projectlog' file can be edited via the *Mail* program, pretty-printed with subject headings, and sorted into chronological order.

OPTIONS

–e Edit 'projectlog' via the *Mail* program.

h Print message headers only.

–h Suppress printing of subject headings.

–p[*low*[*-high*]]
Pretty-print messages in the range *low* to *high.* If *high* is omitted, printing continues to the last message in 'projectlog'.

–s Sort 'projectlog' into chronological order.

FILES

project/projectlog	Log file.

SEE ALSO

Mail(1)
K. Shoens *The Mail Reference Manual*

DIAGNOSTICS

Exit status 0 is normal. Exit status 1 indicates an error.

AUTHOR

Peter J. Nicklin

BUGS

When pretty-printing the 'projectlog' file, *plog* only looks for a 'Subject:' field in the first 5 lines following a 'From' field.

NAME

pman – print project manual

SYNOPSIS

pman [**–P** projectname] [section] topic ...

DESCRIPTION

Pman provides on-line access to the manual belonging to the current working project. If the manual is being printed on a terminal, *pman* pauses after each screenful. Hit a space to continue.

Pman invokes the *man* program to format and print the manual pages corresponding to specified topics. If a section number is given, *pman* looks in that section of the manual for the topics, otherwise all sections of the manual are searched until the topic is found.

The directories containing the project manual must be set up in the project root directory like the '/usr/man' directory hierarchy. In particular, '*project*/man/man1' contains the command manual and '*project*/man/man3' contains the manual for library utilities. Formatted manual entries are created automatically in the corresponding '*project*/man/cat' directories if they exist.

OPTIONS

–P *projectname*
    Specify a project other than the current working project.

EXAMPLES

To print the manual entry for the *pgrep* command from the command section (section 1) of the 'spms' project manual, type:

```
pman pgrep
```

To print the manual entry for the *pgrep* library utility from section 3 of the 'spms' project manual, type:

```
pman 3 pgrep
```

FILES

project/man/man?/*	Unformatted manual pages.
project/man/cat?/*	Formatted on-line manual.

SEE ALSO

man(1), man(7), more(1)

DIAGNOSTICS

Exit status 0 is normal. Exit status 1 indicates an error.

AUTHOR

Peter J. Nicklin

NAME

pmkdir – make a project directory

SYNOPSIS

pmkdir [{+−}**d**] [{+−}N alias] [{+−}**T** type[,type ...]] pdirname ...

DESCRIPTION

Pmkdir creates a directory called *pdirname.* The directory is known as a *project directory.* After the directory has been created, *pmkdir* prompts the user for a line describing its purpose.

If the name of the directory conflicts with an existing project directory, an alternative alias for the project directory may be specified via the **−N** option. However, even if this option is used, the name of the directory will be recognized as a project directory unless it is disguised as a regular pathname. For example, to create another project directory called 'work' with alias 'morework', type:

```
pmkdir  −N morework  ./work
```

Pmkdir may also be used to convert an existing regular directory to a project directory.

Pmkdir requires write permission in the parent directory.

OPTIONS

+d Change the description of an existing project directory.

−d Turn off prompting for the description of a new project directory.

N *alias*
Change the alias of the project directory.

−N *alias*
Specify an alternative alias for a new project directory.

T *type* Add a type label to an existing project directory. If the type label already exists but has a different priority, then it must be removed using the *prmdir* command.

−T *type*
Specify a type label for a new project directory.

FILES

...	Project link directory.
..._temp	Temporary project link directory.

LIMITATIONS

Project directory descriptions can be no longer than 128 characters.

SEE ALSO

mkdir(2), prmdir(1P)

DIAGNOSTICS

The error message, "pmkdir: *project*/... temporarily unavailable", indicates that a '..._temp' temporary project link directory exists. This could be because another user is altering the project link directory, or because a system crash terminated *pmkdir* prematurely. If the latter case, then removing the temporary file will fix the problem.

Exit status 0 is normal. Exit status 1 indicates an error.

AUTHOR

Peter J. Nicklin

BUGS

Directory aliases must not include the characters ':' or '|'.

Type labels must not include the characters ':' or '/'.

NAME

pmv – move or rename files

SYNOPSIS

pmv [–fi] [–] file1 file2

pmv [–fi] [–] file ... dirname

DESCRIPTION

Pmv moves (changes the name of) *file1* to *file2*. If *file2* already exists, it is removed before *file1* is moved. If *file2* has a mode that forbids writing, *pmv* prints the mode and reads the standard input to obtain a line. The move takes place only if the line begins with y. In the second form, one or more *files* are moved to *dirname* with their original file names.

File and *dirname* may be either regular or project pathnames. However, because *pmv* interprets both *file* and *dirname* arguments as project pathnames, if *file* matchs the name of a project directory within the same project, then *pmv* will print the error message 'pmv: can't move project directory *file*', unless *file* is disguised as *./file*.

Pmv blindly supercedes existing files unless the **–i** option is specified.

Pmv refuses to move a file onto itself.

OPTIONS

–f Stands for force. This option overrides any mode restrictions and the **–i** switch.

–i Interactive mode. Whenever a move is to supercede an existing file, the user is prompted by the name of the file followed by a question mark. If answered with a line starting with 'y', the move continues. Any other reply prevents the move from occurring.

– Interpret all the following arguments to *pmv* as file names. This allows file names starting with minus.

SEE ALSO

mv(1)

DIAGNOSTICS

Exit status 0 is normal. Exit status 1 indicates an error.

AUTHOR

Peter J. Nicklin

BUGS

If *file1* and *file2* lie on different file systems, *pmv* must copy the file and delete the original. In this case the owner name becomes that of the copying process and any linking relationship with other files is lost.

NAME
ppd – list project directories

SYNOPSIS
ppd [–1adlmnpqrt] [–T typexpr] [pdirname ...]

DESCRIPTION
Ppd lists the project directories belonging to a project. If *pdirname* is a project root directory, the contents of that project are listed. If *pdirname* is a project directory, the name of that directory is repeated together with any other information requested. When no argument is given, the current project is listed. The output is sorted alphabetically.

OPTIONS

–1	List one entry per line.
–a	List all project directories. Usually '...' and '....' are suppressed.
–d	Print the description of each project directory.
–l	List in long format, giving the full pathname of each project directory.
–m	Mark each project root directory with a trailing '^'.
–n	List the alias corresponding to each project directory.
–p	If *pdirname* is a project root directory, list only its name and not its contents. This option is used often with **–d** and **–t** to get the description or type labels of a project root directory.
–q	Quiet mode. Do not print subproject titles. Mostly used with a combination of **–l** and **–r** to get a list of pathnames for a project hierarchy.
–r	Recursively list subprojects.
–t	Print the type labels of each project directory.
–T *typexpr*	Only list project directories corresponding to boolean type label expression, *typexpr*.

DIAGNOSTICS
Exit status 0 is normal. Exit status 1 indicates an error.

AUTHOR
Peter J. Nicklin

NAME

prmdir – remove a project directory

SYNOPSIS

prmdir [**–fru**] [{+–}T type[,type ...]] pdirname ...

DESCRIPTION

Prmdir deletes a project directory called *pdirname*. The directory must be empty.

If the **–r** option is specified, *prmdir* recursively deletes the entire contents of a project directory, and the directory itself. The user is asked to confirm the generated *rm –r* command before the directory is deleted. Subdirectories that are project root directories must be removed using *rmproject* before attempting to remove *pdirname*. Write permission is required in all subdirectories.

Prmdir may also be used to convert an existing project directory to a regular directory using the **–u** option.

OPTIONS

–f Stands for force. No questions are asked. This option overrides any mode restrictions.

–r Recursively remove project directories.

–u Undefine a project directory and convert it to a regular directory.

–T *type*
Remove a type label from a project directory.

FILES

...	Project link directory.
..._temp	Temporary project link directory.

SEE ALSO

pmkdir(1P), rm(1), rmdir(1), rmproject(1P)

DIAGNOSTICS

The error message, "prmdir: *project*/... temporarily unavailable", indicates that a '..._temp' temporary project link directory exists. This could be because another user is altering the project link directory, or because a system crash terminated *prmdir* prematurely. If the latter case, then removing the temporary file will fix the problem.

Exit status 0 is normal. Exit status 1 indicates an error.

AUTHOR

Peter J. Nicklin

BUGS

If a project directory has already been removed by the *rmdir* or *rm -r* commands, that directory must be recreated using *mkdir* before *prmdir* will remove the directory from the project.

NAME

ptest – test a project module

SYNOPSIS

ptest [–d] [–F template] **[–P** projectname] [module ...]

DESCRIPTION

Ptest does regression testing on project *modules.* Individual modules can be selected for testing, or if *ptest* is given without *module* arguments, all of the modules in a project are tested. Modules can be programs, libraries, or functions.

For each module to be tested there must exist an archive containing a test program, input data, and test results. *Ptest* tests a module by extracting the archive, compiling the test program, executing the test, and comparing the output with expected test results using the *diff* program:

```
T*test* < I*test* | diff - O*test*
```

where *test* is the name of the module test. If the test output differs from expected results, the test fails and error diagnostics are saved in a file named 'E*test*'.

Compiling The Test Program

If an archive contains a compilable test program, *ptest* uses *mkmf* to edit the file name of the test program into a makefile which is then used by *make* to compile the program. If there are other source code file names in the current working directory, these are also included in the makefile. This mechanism allows experimental versions of functions to be tested since they will override any other occurrences of the functions (e.g. in a library).

Creating Test Archives

Archives containing module tests reside in the 'test' directory located in the project root directory. Each archive must be created by the *ar* command and given the name *test*.a where *test* is the name of a module test. These archives include the following files:

(a) 'T*test.lang*' test program source code. The file name suffix, *lang,* identifies the programming language in which the test program is written.

(b) 'T*test*.sh' shell command script. 'T*test*.sh' executes a test and compares the test output with expected results stored in the archive. If the output matchs expected results, 'T*test*.sh' returns with exit code 0. If the results differ, the exit code should be nonzero.

Note: the shell command script must be made executable by doing:

```
chmod +x T*test*.sh
```

If a 'T*test*.sh' file is not found in the archive, *ptest* executes the following command:

```
T*test* < I*test* | diff - O*test*
```

or if there is no 'I*test*' data file:

```
T*test* | diff - O*test*
```

(c) 'I*test*' data file for test input.

(d) 'O*test*' data file for validated test output. The output from a module test will be compared against the information in this file. If no output is expected, this file should be zero size.

The files 'T*test.lang*', 'T*test*.sh', and 'I*test*' may be omitted from the archive.

OPTIONS

–d Leave the files that have been extracted from an archive, in the current directory for debugging purposes.

−F *template*
: Specify an alternative makefile *template* file name. The default file name is 't.Makefile'.

−P *projectname*
: Specify a project other than the current working project.

FILES

/usr/new/lib/t.Makefile	Standard test program makefile template.
project/lib/t.Makefile	User-defined test program makefile template.
project/test/*.a	Project test archives.
test.a	Module test archive.
E*test*	Module error diagnostic file.
I*test*	Module input data file.
O*test*	Module validated output data file.
T*test*	Compiled module test program.
T*test*.sh	Module shell command script.
T_makefile	Test program makefile.

SEE ALSO

ar(1), chmod(1), diff(1), make(1), mkmf(1P)

DIAGNOSTICS

Exit status 0 if the test succeeds. Exit status 1 if it fails.

AUTHOR

Peter J. Nicklin

BUGS

Since the PATH environment variable governs the order in which directories are searched for executable commands, the production version of a command may be used instead of an experimental version.

Any files beginning with 'T' (except those with suffix '.sh') are deleted prior to each test.

NAME
pwp – print working project name

SYNOPSIS
pwp [–l]

DESCRIPTION
Pwp prints the name of the current working project.

OPTIONS
–l List in long format, giving the full pathname of the project root directory.

DIAGNOSTICS
The error message "pwp: ./....: No such file or directory", may occur if a project link directory is missing, or the parent project '....' cannot be found. The latter case can happen if the project has been moved recently.

Exit status 0 is normal. Exit status 1 indicates an error.

AUTHOR
Peter J. Nicklin

NAME

rmproject – remove a project root directory

SYNOPSIS

rmproject [–fru] [{+–}T type[,type ...]] projectname ...

DESCRIPTION

Rmproject deletes a project called *projectname.* The project must be empty.

If the **–r** option is specified, *rmproject* recursively deletes the entire contents of a project root directory, and the directory itself. The user is asked to confirm the generated *rm -r* command before the project is deleted. Subdirectories that are project root directories must be removed using *rmproject* prior to removing *projectname.* Write permission is required in all subdirectories.

Rmproject may also be used to convert an existing project root directory to a regular directory using the **–u** option. However, subdirectories that are project root directories must be undefined using *rmproject -u* prior to undefining *projectname.*

OPTIONS

–f Stands for force. No questions are asked. This option overrides any mode restrictions.

–r Recursively remove project directories.

–u Undefine a project root directory and convert it to a regular directory.

–T *type*
Remove a type label from a project root directory.

FILES

...	Project link directory.
..._temp	Temporary project link directory.

SEE ALSO

mkproject(1P), rm(1), rmdir(1)

DIAGNOSTICS

The error message, "rmproject: *project/...* temporarily unavailable", indicates that a '..._temp' temporary project link directory exists. This could be because another user is altering the project link directory, or because a system crash terminated *rmproject* prematurely. If the latter case, then removing the temporary file will fix the problem.

When attempting to remove a project, error message "rmproject: *project/...*: No such file or directory" followed by error message "rmproject: force removal by typing 'rmproject -F projectname' " indicates that a project link directory is missing. In this case, *projectname* must be an absolute project pathname.

When attempting to undefine a project, error message "rmproject: *project/...*: No such file or directory" followed by error message "rmproject: force conversion by typing 'rmproject -uF projectname' " indicates that a project link directory is missing. In this case, *projectname* must be an absolute project pathname.

Exit status 0 is normal. Exit status 1 indicates an error.

AUTHOR

Peter J. Nicklin

BUGS

There is no restriction on overlapping project hierarchies. A project root directory that is part of another project hierarchy will be removed without complaint.

If a project root directory has already been removed by the *rmdir* or *rm -r* commands, that directory must be recreated using *mkdir* before *rmproject* will remove the project.

SUMACC
MAC INTOSH CROSS DEVELOPMENT SYSTEM

William Croft, Stanford

NAME

as68 – .a68 -> .b assembler component of cc68

SYNOPSIS

as68 [-godspel] filename

DESCRIPTION

As68 is the 68000 assembler. The input is taken from filename.a68, if present, otherwise from filename. The output is sent to filename.b. More than one input file can be specified, but only a single output is generated. The available flags are

-g Undefined symbols are automatically declared global for later resolution by the loader.

-o filename
Direct output to filename.

-d Print info helpful for debugging the assembler

-s Put symbol table in list.out (relocatable values only)

-p Print listing on stdout

-e External symbols only in output

-l produces a listing, filename.list

FILES

/usr/sun/a68 /usr/bin/as68 /usr/sun/doc/a68opcodes

SEE ALSO

cc68 (1), pc68(1), ld68 (1).

NAME

cc68 – C compiler for the MC68000

SYNOPSIS

cc68 [option] ... file ...

DESCRIPTION

Cc68 is the UNIX C compiler modified for the MC68000. *Cc68* is a flexible program for translating between various types of files. The types catered for in order of appearance during translation are '.c' (C source files), '.s' (assembly language files), '.b' (relocatable binary files), 'b.out' (absolute binary files), '.r' (byte-reversed files, cf. *rev68(1)*). and '.dl' (Macsbug download format, cf. *dl68(1)*).

Arguments to cc68 are either flags or input files. The type of an input file is normally determined by its suffix. When an argument to cc68 is not a flag and has a suffix different from any of the above suffixes, it is assumed to be of one of the types '.c', '.b', or 'b.out', namely the latest of these three consistent with the type of the output (e.g. if the output type were '.s' or '.b' then the input would have to be '.c'). If it has no suffix it is assumed to be of type 'b.out'.

Translation proceeds as follows. Each '.c' and '.s' program is translated to a '.b' relocatable using cpp, ccom68, and as68 as necessary. Then all .b files including those produced by translation are link edited into the one file, called 'b.out'. If the only input file was a single '.c' program then the '.b' file is deleted, otherwise all '.b' files are preserved.

The amount of processing performed by cc68 may be decreased or increased with some of the options. The -S option takes translation no further than '.s' files, i.e. only cpp and ccom68 are applied. The -c option takes translation up to '.b' files, omitting the link editing and not deleting any '.b' files. The -d option goes beyond 'b.out' to produce a '.dl' file (using dl68) that may be downloaded by the Motorola MACSBUG monitor and the Sun1 monitor. The -r option similarly goes beyond 'b.out' to produce a '.r' file (using rev68) that may be loaded directly by 68000 code based on ld68. Both -d and -r may be used together.

The output may be named explicitly with the -o option; the output file's name should follow -o. Otherwise the name is 'b.out' in the normal case, or 'filename.dl' for the -d option, or 'filename.r' for the -r option, where 'filename' is the first '.c', '.s', or '.b' file named as an input. If the input is not in any of those three categories, the names 'd.out' and 'r.out' are used respectively for -d and -r.

The version of the target machine may be given as the flag -v*n* where *n* is the version. The only recognized version at present is -vm, "Version Macsbug." The effect of giving the -vm flag is to add /usr/sun/dm/include to the include directories for cpp, to add /usr/sun/dm/lib as a library in which to look for -lx libraries, and to load the symbol table if any into the region starting at 0x6BA.

The file /usr/sun/lib/crt0.b is passed to ld68, ahead of all other .b files. This has the effect of defining the symbol _start to be at the text origin and having a routine that performs necessary initialization, enters main, and exits cleanly to the monitor.

The following options are interpreted by *cc*. See *ld68*(1) for load-time options.

–d Produce a .dl file suitable for downloading with the MACSBUG monitor of the Motorola Design Module, cf. *dl68*(1).

–r Produce a .r file suitable for direct loading by the 68000, cf. *rev68*(1).

–c Suppress the loading phase of the compilation, and force an object file to be produced even if only one program is compiled.

–w Suppress warning diagnostics. **[Note: may not work.]**

-O Invoke an object-code improver.

-S Compile the named C programs, and leave the assembler-language output on corresponding files suffixed '.s'.

-E Run only the macro preprocessor on the named C programs, and send the result to the standard output.

-L Produce an assembly listing for each source file, with the suffixes changed to ".ls".

-R Preserve relocation commands in b.out.

-C prevent the macro preprocessor from eliding comments.

-V Link for a V kernel environment. This is equivalent to specifying **-i/usr/sun/lib/teamroot.b -T 10000** and **-lV at the end.**

-m Link for a Macintosh environment. This is equivalent to specifying **-i/usr/sun/lib/crtmac.b -T 0 -e _start -r -d** and **-lmac -lc** at the end.

-o *output*
Name the final output file *output*. If this option is used and the file 'b.out' already exists it will be left undisturbed.

-l*x* Include libx.a as a library ld68 should search in for undefined functions. x may be more than one letter, as in -lpup.

-T *org* Org specifies in hexadecimal where to begin loading the program.

-e *entrypoint*
Entrypoint specifies where to begin execution.

-D*name=def*
-D*name*
Define the *name* to the preprocessor, as if by '#define'. If no definition is given, the name is defined as "1".

-U*name*
Remove any initial definition of *name*.

-I*dir* '#include' files whose names do not begin with '/' are always sought first in the directory of the *file* argument, then in directories named in **-I** options, then in directories on a standard list. The standard list is (in order of search) */usr/sun/include* and */usr/include*.

-B*string* Find substitute compiler passes in the files named *string* with the suffixes cpp, ccom and c2. If *string* is empty, use a standard backup version. **[Which doesn't work!]**

-t[p012]
Find only the designated compiler passes in the files whose names are constructed by a **-B** option. In the absence of a **-B** option, the *string* is taken to be '/usr/c/'.

—x By default, *cc68* passes a -x flag to *ld68*, in order to suppress local symbols from the final symbol table. The —x flag inhibits this default.

Other arguments are taken to be either loader option arguments, or C-compatible object programs, typically produced by an earlier *cc68* run, or perhaps libraries of C-compatible routines. These programs, together with the results of any compilations specified, are loaded (in the order given) to produce an executable program with name **b.out.**

FILES

file.c	input file
file.b	object file
b.out	loaded output
/tmp/ctm?	temporary

/lib/cpp	preprocessor
/usr/sun/c68/comp	compiler
/usr/sun/c68/o68	optional optimizer
/usr/sun/lib/crt0.b	runtime startoff
/usr/sun/lib/libc.a	standard library, see (3)
/usr/sun/include	
/usr/include	standard directories for '#include' files

SEE ALSO

B. W. Kernighan and D. M. Ritchie, *The C Programming Language,* Prentice-Hall, 1978
B. W. Kernighan, *Programming in C—a tutorial*
D. M. Ritchie, *C Reference Manual*
ld68(1)

DIAGNOSTICS

The diagnostics produced by C itself are intended to be self-explanatory. Occasional messages may be produced by the assembler or loader.

BUGS

This is hacked up from *cc*(1), and probably could be improved.

NAME

ddt68, fddt68 – symbolic debugger for 68000

SYNOPSIS

fddt68 b.out
cc68 ... -lddt (Sun)

DESCRIPTION

fddt68 is a symbolic disassembler for b.out files created by the 68000 linker (ld68). Its main purpose is to allow testing of ddt logic in a more hospitable environment than the 68000. It also gives a way of inspecting the assembly language form of a program without having to produce a .s file. In addition it gives a check on the operation of as68 and ld68. It is called by typing:

fddt68 *filename*

on the Vax.

ddt68 is a symbolic debugger for the 68000. It is loaded at link edit time with the cc68 flag -lddt. On starting a program with ddt loaded the user will be at the ddt command level. Breakpoints may be set, and the program started, using the commands described below.

COMMANDS

ddt68 recognizes the following commands (*$* is used for *<esc>*):

expression /

expression \

open the location at *expression* and display the contents in the current typeout mode. The user may then optionally type an expression, whose value replaces the contents of the open location. Finally the location is *closed* by typing one of *return* (to return to *ddt*'s main command loop), / (to open the next location), or \ (to open the previous location).

*expression***$g**

go - plant any breakpoints set with the *$b* command, load the registers, and start execution at *expression*. If *expression* is unspecified or zero, execution resumes starting from the current value of *$pc* (normally the point where the program was last interrupted).

*expression***$x**

execute the next *expression* instructions, starting from the current value of *$pc* and printing out all executed instructions. If *expression* is omitted, 1 is assumed.

*expression***$$x**

same as above except execute subroutine calls and traps as single instructions, i.e. do not descend into the called subroutine.

*expression***$p**

proceed - like *go* with no argument, except that if we are presently at a breakpoint then *expression* counts the number of times to pass this breakpoint before breaking. *1$p* is synonymous with *$g*.

*expression$bno***b**

set breakpoint *bno* (in the range 1-9) at *expression*. If *bno* is omitted the first unused breakpoint number is assigned (the commonest usage). If *expression* is 0 the named breakpoint is cleared, or if there is no named breakpoint (*bno* is omitted) all breakpoints are cleared. If *expression* is omitted all breakpoints are printed, whether or not *bno* is present.

$*rspec*/

$*rspec*\

examine register *rspec* where *rspec* is one of:

d0-d7	data registers 0-7
a0-a7	address registers 0-7
fp	frame pointer (synonym for *a6*)
sp	stack pointer (synonym for *a7*)
pc	program counter
sr	status register

expression$**=**

type out *expression* in current output radix.

lowlimit<*highlimit*>*pattern*?

search for *pattern* in the range *lowlimit* (inclusive) to *highlimit* (exclusive). The pattern is interpreted as an object of the type in force as the current typeout mode, with instructions and strings being treated as 2-byte words. Objects are assumed to be aligned on word (2-byte) boundaries except for 1-byte types and strings which are aligned on byte boundaries. A mask (set with the following command) determines how much of the pattern is significant in the search, except that if the pattern is a string constant a separate mask matched to the length of the string is used. The three arguments to the search command are sticky; thus if *lowlimit*< (resp. *highlimit*>) is omitted, the most recent lowlimit (resp. highlimit) applies. While *pattern* may be omitted, the final ? may not be omitted.

expression$**m**

set the search mask to *expression*. *-1$m* forces a complete match, *f$m* checks only the low order 4 bits, *0$m* will make the search pattern match anything.

base$**ir** set input radix to *base*. (Note *10$i* can never change the radix.) If *base* is omitted hexadecimal is assumed.

base$**or**

set output radix to *base*. If *base* is omitted hexadecimal is assumed.

$*type*t temporarily set typeout mode to *type* where *type* is one of:

<space>	deduce type from type of nearest symbol
c	type out bytes as ascii characters.
h	type out bytes in current output radix.
w	type out words in current radix.
l	type out longs in current radix.
s	type out strings in current radix. (In this mode new values cannot be entered.)
i	type out as 68000 symbolic instructions. (In this mode only the first two bytes of the opened location may be changed; the new value is typed in as a numeric expression rather than as a symbolic instruction.)

The new typeout mode stays in effect until a *return* is typed.

$$*type*t

permanently set typeout mode to *type*.

An *expression* is composed of symbols, numeric constants, string constants, and the operators +, -, and | representing 2's complement addition, subtraction, and inclusive bitwise or. Symbols are delimited by operators or <*esc*>. A string constant has from 1 to 4 characters which are packed right justified into one long to form a numeric constant; thus "did"=646A64. String constants are particularly useful in conjunction witht the search command for searching for a string. The single character . (dot) as a symbol on its own represents the address of the currently open memory location. All operations are carried out using 32 bit arithmetic and evaluated strictly left to right.

AUTHORS

Jim Lawson and Vaughan Pratt

NAME

dl68 – b.out -> .dl downloader component of cc68

SYNOPSIS

dl68 [-T -v -o -s] filename

DESCRIPTION

Dl68 is a downloader for the Motorola 68000 Design Module. It takes its input, a b.out format file, from filename and in the absence of the -o option sends its output to stdout.

If there are any symbols these are loaded, starting at 0x6BA on vm (the Design Module) or 0x1F000 on v1 (the Sun1 prototype). The start and end of the symbol table are stored at 0x570 and 0x574 respectively on either board.

The options are:

–T *textorigin*
specifies where the text (code) is to be loaded.

–v*n* specifies the board version. Default is v1 (Sun1 prototype). vm denotes the Motorola Design Module.

–o *filename*
specifies the output file. Defaults to stdout.

–s*DE* specifies the *data/end* record types to generate. The default is s28, 24 bit addresses. The s19 format, 16 bit addresses, is used by the Data I/O programmers.

FILES

/usr/sun/ld68/down.c /usr/bin/dl68

NAME

ld68 – .b -> b.out linker for the MC68000

SYNOPSIS

ld68 [option] ... file ...

DESCRIPTION

Ld68 combines several object programs into one, resolves external references, and searches libraries. In the simplest case several object *files* are given, and *ld68* combines them, producing an object module which can be either executed or become the input for a further *ld68* run. (In the latter case, the **-r** option must be given to preserve the relocation bits.) The output of *ld68* is left on **b.out**. This file is made executable only if no errors occurred during the load.

The argument routines are concatenated in the order specified.

The entry point of the output is determined by the first applicable item of the following list: the **-e** option if given, the value of the symbol _start if defined, or the text origin (first instruction).

If any argument is a library, it is searched exactly once at the point it is encountered in the argument list. Only those routines defining an unresolved external reference are loaded. If a routine from a library references another routine in the library, the referenced routine must appear after the referencing routine in the library. Thus the order of programs within libraries may be important.

The symbols '_etext', '_edata' and '_end' ('etext', 'edata' and 'end' in C) are reserved, and if referred to, are set to the first location above the program, the first location above initialized data, and the first location above all data respectively. It is erroneous to define these symbols.

Ld68 understands several options. Except for **-l**, they should appear before the file names.

-D Take the next argument as a decimal number and pad the data segment with zero bytes to the indicated length.

-d Force definition of common storage even if the **-r** flag is present.

-e The following argument is taken to be the name of the entry point of the loaded program; location 0x1000 is the default.

-f Fold case on identifiers. That is, upper and lower case letters are not distinguished. Used to link with Pascal routines, for example.

-l*x* This option is an abbreviation for the library name '/usr/sun/lib/lib*x*.a', where *x* is a string. A library is searched when its name is encountered, so the placement of a **-l** is significant.

-v*x* This denotes board version *x* which may at present only be 'm' for Motorola Design Module. The default board version is the Sun1 prototype, v1.

-M Create a human-readable list of symbols in "sym.out".

-n Arrange (by giving the output file a 0410 "magic number") that when the output file is executed, the text portion will be read-only and shared among all users executing the file. This involves moving the data areas up to the first possible 64K byte boundary following the end of the text (not really useful yet).

-o The *name* argument after **-o** is used as the name of the *ld68* output file, instead of **b.out.**

-q Quicksort symbols in **b.out** in ascending numerical order.

-r Generate relocation bits in the output file so that it can be the subject of another *ld68* run. This flag also prevents final definitions from being given to common symbols, and suppresses the 'undefined symbol' diagnostics.

-S 'Strip' the output by removing all symbols except locals and globals.

-s 'Strip' the output, that is, remove the symbol table and relocation bits to save space (but impair the usefulness of the debuggers). This information can also be removed by *strip*(1).

-T The next argument is a hexadecimal number which sets the text segment origin. The default origin is 0x1000. If you intend to use the output as input to another run of ld68, you must specify -T 0.

-B The next argument is a hexadecimal number which sets the common/bss segment origin. The default origin is immediately after the data segment.

-u Take the following argument as a symbol and enter it as undefined in the symbol table. This is useful for loading wholly from a library, since initially the symbol table is empty and an unresolved reference is needed to force the loading of the first routine.

-X Save local symbols except for those whose names begin with 'L'. This option is used by *cc*(1) to discard internally-generated labels while retaining symbols local to routines.

-x Do not preserve local (non-.globl) symbols in the output symbol table; only enter external symbols. This option saves some space in the output file.

FILES

/usr/sun/lib/lib*.a	libraries
b.out	output file

SEE ALSO

ar(1), cc68(1), a68(1)

BUGS

The b.out format header does not contain any indication of the text segment origin, so if you specify something other than the default origin -T 1000, you will have to remember this value and specify it again to dl68 when you download. The standard Sun monitor cannot netload files with origins other than 1000, so you must either use dl68 or write a special loader for such programs.

NAME

macget – receive file from macintosh via modem7/macterminal

SYNOPSIS

macget [**-rdu**] [file]

DESCRIPTION

Macget receives a file from a Macintosh running MacTerminal. The File Transfer settings should specify the "Modem7" transfer method and a "MacTerminal" remote system. This program is designed for use with the 0.5 Beta and newer versions of MacTerminal, but includes a compatibility option for the older -0.15X Almost-Alpha version.

To use this program, log into the unix system using MacTerminal, start macget with the desired options, select "Send File..." from the "File" menu, and open the file you wish to send. If MacTerminal is properly configured, it will put up an indicator showing how much of the file has been transfered. Several Control-X's may be used to force macget to give up if the transfer fails.

The optional *file* parameter specifies the name to use when creating the unix files, otherwise the Mac file name is used (with spaces converted to underscores).

If none of the **-rdu** flags are specified, *macget* receives three files from the Mac: *file*.**info**, *file*.**data**, and *file*.**rsrc**. This mode is useful for storing Mac files so they can be restored later using *macput*.

The **-r** flag specifies *resource* mode. Only *file*.**rsrc** will be created, from the Mac file's resource fork.

The **-d** flag specifies *data* mode. Only *file*.**data** will be created, containing the data fork of the Mac file.

The **-u** flag requests *unix* mode, in which carriage returns are converted into unix newline characters, and the unix file *file*.**text** is created. A file saved from Mac applications as "text only" can be transfered using this option to convert it to a normal unix text file.

The **-o** flag specifies "old" (version -0.15X) MacTerminal compatibility mode. You must manually disable XON/XOFF flow control in this version to perform file transfer; this is done automatically in the newer versions.

SEE ALSO

macput(local)

BUGS

Doesn't work over flow controlled communication lines, or when using rlogin.

AUTHOR

Dave Johnson, Brown 7/31/84

NAME

macput – send file to macintosh via modem7/macterminal

SYNOPSIS

macput file
macput [**–rdu**] file [**–t** type] [**–a** author] [**–n** name]

DESCRIPTION

Macput sends a file to a Macintosh running MacTerminal. The File Transfer settings should specify the "Modem7" transfer method and a "MacTerminal" remote system. This program is designed for use with the 0.5 Beta and newer versions of MacTerminal, but includes a compatibility option for the older -0.15X Almost-Alpha version.

To use this program, log into the unix system using MacTerminal, and run macput specifying the desired options and one file to be sent. If MacTerminal is properly configured, it will recognize that a file is arriving on the serial line and put up an indicator showing how much of the file has been sent. Several Control-X's may be used to force macput to give up if the transfer fails.

If none of the **–rdu** flags are specified, *macput* sends three files to the mac: *file*.**info**, *file*.**data**, and *file*.**rsrc**. This is useful for returning files to the mac which were stored using macget.

The **–r** flag specifies *resource* mode. Either *file*.**rsrc** or *file* will be sent to the Mac, along with a forged **.info** file and an empty **.data** file. The file sent becomes the resource fork of the Mac file.

The **–d** flag specifies *data* mode. Either *file*.**data** , *file*.**text** or *file* will be sent to the Mac, along with a forged **.info** file and an empty **.rsrc** file. The file sent becomes the data fork of the Mac file.

The **–u** flag requests *unix* mode, which is the same as *data* mode except unix newline characters are converted into carriage returns. Human-readable unix text files sent to the Mac using this option will be compatible with applications which expect "text only" files.

The **–o** flag specifies "old" (version -0.15X) MacTerminal compatibility mode. You must manually disable XON/XOFF flow control in this version to perform file transfer; this is done automatically in the newer versions.

The remaining options serve to override the default file type, author, and file name to be used on the Mac. The default type and author for *resource* mode are "APPL" and "CCOM". *data* mode defaults are "TEXT", "????", and *unix* mode defaults are "TEXT" and "MACA".

SEE ALSO

macget(local)

BUGS

Doesn't work over flow controlled communication lines, or when using rlogin.

Doesn't set the bundle bit on resource files, to incorporate any icons into the Desk Top. Use setfile to set the bundle bit.

FEATURES

Properly initializes the Creation Date.

AUTHOR

Dave Johnson, Brown 7/31/84

NAME

pc68 – Pascal compiler for the MC68000

SYNOPSIS

pc68 [option] name ...

DESCRIPTION

Pc68 is the version of the portable Pascal* compiler that generates code for the MC68000. *Pc68* is a flexible program for translating between various types of files. The types catered for in order of appearance during translation are '.p' (Pascal source files), '.a68' or '.s' (assembly language files), '.b' (relocatable binary files), 'b.out' (absolute binary files), '.r' (byte-reversed files, cf. *rev68(1)*). and '.dl' (Macsbug download format, cf. *dl68(1)*).

Arguments to pc68 are either flags or input files. The type of an input file is normally determined by its suffix. When an argument to pc68 is not a flag and has none of the above suffixes, it is assumed to be of one of the types '.p', '.b', or 'b.out', namely the latest of these three consistent with the type of the output (e.g. if the output type were '.s' or '.b' then the input would have to be '.p').

Translation proceeds as follows. Each '.p' and '.s' program is translated to a '.b' relocatable using upas68, ugen68, and as68 as necessary. Then all .b files including those produced by translation are link edited into the one file, called 'b.out'. If the only input file was a single '.p' program then the '.b' file is deleted, otherwise all '.b' files are preserved.

The amount of processing performed by cc68 may be decreased or increased with some of the options. The -S option takes translation no further than '.s' files, i.e. only upas68 and ugen68 are applied. The -c option takes translation up to '.b' files, omitting the link-editing and not deleting any '.b' files. The -d option goes beyond 'b.out' to produce a '.dl' file (using dl68) that may be downloaded by the Motorola MACSBUG monitor and the Sun1 monitor. The -r option similarly goes beyond 'b.out' to produce a '.r' file (using rev68) that may be loaded directly by 68000 code based on ld68. Both -d and -r may be used together.

The output may be named explicitly with the -o option; the output file's name should follow -o. Otherwise the name is 'b.out' in the normal case, or 'filename.dl' for the -d option, or 'filename.r' for the -r option, where 'filename' is the first '.p', '.a68', '.s', or '.b' file named as an input. If the input is not in any of those three categories, the names 'd.out' and 'r.out' are used respectively for -d and -r.

The version of the target machine may be given as the flag –v*n* where *n* is the version. -vm is "Version Macsbug." -vV means to run under the Vkernal. This is pretty much a hack: File I/O is not supported and you can't refer to C routines containing upper-case letters in their names. It's also rather minimally tested.

A complete list of options interpreted by pc68 follows:

Pass

flag to the compiler. See the SOURCE FLAGS section below.

–c Suppress loading and produce '.b' file(s) from source file(s).

–g Have the compiler produce additional symbol table information for *pcdb68* (not implemented).

–e entrypoint
Entrypoint specifies where to begin execution.

–o output
Name the final output file *output* instead of *b.out.*

–s Accept standard Pascal only; non-standard constructs cause warning diagnostics (not implemented – see internally controlled options).

-v n Use the 'n' version of the runtime support.

-w Suppress warning messages (not implemented).

—x Suppress passing the '-x' flag to the loader, retaining local symbols.

-E Run only the preprocessor (not implemented).

-L Make an assembly listing in filename.ls for each file assembled.

-O Invoke an object-code improver (not implemented).

-R Preserve relocation information in b.out.

-S Compile the named program, and leave the assembler-language output on the corresponding file suffixed '.s'. (No '.b' is created.).

-T org
Org specifies in hexadecimal where to begin loading the program.

-V Show the various stages of the compilation by printing images of the processes forked off to perform the actual work of the compilation.

-U Save the ucode associated with filname.p in filename.u (and filename.z, depending on the -W option).

-W Invoke the global ucode-to-ucode optimizer. If -U option active, generates filename.z.

-P Save all intermediate files. Most useful in conjuntion with -V (so that it is possible to find the intermediates).

Other arguments are taken to be loader option arguments, perhaps libraries of *pc68* compatible routines.

SEPARATE COMPILATION

Object files created by other language processors may be loaded together with object files created by *pc68*. Calling conventions are as in C, with **var** parameters and arrays passed by address. Don't pass structures except by VAR (pointer) if you call C, since here pc68 and cc68 differ. As a convenience, string constants are followed by a zero byte, so that you can use them as C strings when calling C routines.

To refer to a subroutine defined in a separate module, it must be declared. This follows the same syntax as **forward** declarations, except that the keyword **FORWARD** is replaced by EXTERN .

A file of subroutines is similar to a program except that there is no main program, and the **program** statement at the beginning of the file is replaced by a statement:

```
  MODULE modulename;
```

The 'end;' of the last function in the file is followed by a period - there is no main program block.
The modulename will become significant in Pascal*. Note that in identifiers (such as subprogram names) upper case is changed to lower case, and the linker is asked to ignore case.

OPENING FILES

To open a file for both input and output, use the standard procedure REVISE, which is analogous with RESET and REWRITE. NOT TESTED.

You can read and write files on machines which run a Leaf server. To open a file for reading do:

```
  reset(file,'[hostname:username:password]filename');
```

The same syntax applies to rewrite. You can of course also use a Pascal string variable.

Terminating spaces in hostname, username and password are ignored. (This should make it easier for a program to construct the appropriate filename string.)

You can leave out fields (or the entire second parameter), and the program will assume you want the same as before. If there is no "before", it will ask you.

Reset, Rewrite and Revise may have an optional third parameter, which is a string of switches. E.g.:
    Reset(Input,'data1.txt','Nofilter;Prompt:"Try again!"');

Standard switches are:
- Prompt: The string is used as a prompt (interactive systems only). If a file name is NOT given, this prompt is used to get the file name from the user. If one IS given (like in the example above), the prompt is used to get another file name from the user if the file can't be opened.
- Default: The string is used are used as a default file name, which is used if the user types a carriage return in response to the prompt.
- Standard: If Reset, the standard input file is used. If Rewrite, the standard output is used.
- Nofilter: (Reset, Revise only.) Normally a text file is 'filtered' by the runtimes so that it conforms to the standard Pascal definition of a text file. Most notably, any end-of-line characters are changed into one space. The inclusion of Nofilter causes all characters to be passed through exactly as they appear in the text file. Eoln, Eopage and Readln still work as for standard files.

EXTENSIONS TO READ AND WRITE

For all field widths (if there are two field-width-type parameters, the first one only), a negative value will mean that the value written will be left-aligned instead of right-aligned. For string variables, if Abs(Fieldwidth) < Length, then the last Length - Abs(Fieldwidth) characters of the string will be written.

Variables of enumerated types may be read and written. The field width is interpreted the same as for strings. Enumerated constant names are uppercased when they are read in.

Sets of readable and writeable types may also be read and written. They appear exactly as set constants appear in Pascal programs. The field width is interpreted for each element the same as it would be for the set element type.

Integers may be written in other bases beside base 10 by including an optional field-width-type parameter, which may be anywhere from 1..16. The field width is the same as for base 10. Integers may also be read from a file in other than base 10, by including a field-width-type parameter in the call to Read or Readln.

Real numbers may have a capital "E" as well as the standard small "e" in the exponent part.

MORE ABOUT INPUT-OUTPUT

Lazy lokahead is used for text files, so that terminal input works reasonably.

The procedure Eopage is true iff a page marker has just been read, and the corresponding space in now in the file buffer.

Random-access in files is done with the standard procedure

seek (File, N);
This positions the file so that the next read/write will apply to component no. N of the file.

To close a file immediately do: close(file);

Function Filesize (var Filevar: Anyfile): 0..Maxint returns the current number of components in a file.

Function Curpos (var Filevar: Anyfile): 0..Maxint: Returns the current file position.

Procedure Filepos (var Filevar: Text; var Pagenum, Linenum, Charnum: 0..Maxint): Returns page, line number, and column number of the next character that will be read from the file (must be open for input). Does not work for random access.

TIME AND DATE ROUTINES

Clock -returns milli-seconds since the monitor was booted.

The following routines routines don't work if you want to run stand-alone, but need an operating system (V or Unix).

Ptime -returns (in theory) milli-seconds since midnight.
(under V, actually returns seconds*1000)

Pdate(day, month, year) -set day, month & year (say 1982).

Time(string) -sets string to 'HH:MM:SS'

Date(string) -sets string to 'MM/DD/YY'.
For both time and date, the string is a packed array [1..n] of char, where n>=8. (Any overflow is set to spaces.)

OTHER EXTENSIONS AND FEATURES

An "others" label in as CASE statement, indicates a default case.

To include a file as part of the program source do:
INCLUDE 'filename';
This is especially useful for declarations for seperately compiled modules.

Records declared as "packed" will be packed down to individual bits; however elements of packed arrays are at least a byte.

Function Min (X,Y: T): T -- returns the minimum of two arguments, which may be of any ordinal or real type.

Function Max (X,Y: T): T -- returns the maximum of two arguments.

Procedure Halt (Exitcode: Integer): Causes abnormal termaination of a program. Passes a system-dependent exit code to the operating system.

The comment pairs '{ }' and '(* *) match independently, allowing limited nesting of comments.

SOURCE FLAGS

These flags can be passed to the compiler either at the command level when invoking pc68, or as comments within the program. A sample option line is a comment with # as its first character:
Sample option line: (*#g+,tdpy 1,tchk 1,U-8 *)
WARNING: Only (* *)-style comments will work; {#...} is ignored!
Sample command line: pc68 file.p #g:+ #tdpy:1 #tchk:1 #U:-8

Switch	Meaning (Note that the default value is shown)
B+	Bounds and nil pointer checking
C+	Print ucode
D-	Load with debugger
E+	Emit source code (for system debugging)
G-	Write error messages only to listing file
L-	Write full listing
I16	Number of characters of identifiers that are considered significant
O-	Emit optimizer-compatible code
P-	Keep execution profile
R0	Put up to N local variables in (data) registers (Register allocation should be done by the optimizer.)
S-	Accept standard Pascal only
T—	Code generator options
U+	Leave procedure names exactly as is
V32	Number of bits (16 or 32) to allocate for 'Integer'.
Wn	PRINT WARNINGS FOR:
W1	unused variables, types, procs, etc.
W10	nested comments
Z—	Optimizer switches

FILES

file.p	pascal source files
file.b	binary files
file.a68	assembler files
file.s	assembler files
file.ls	assembler listing
file.err	pascal listing

BUGS

Displacements off a frame pointer is limited to 16 bits signed, so very large locally-defined arrays will crash.

Some attempted bogus conversions (e.g. structure to real) aren't detected by the front end, and result in messages about 'Illegal CVT datatypes' from the code generator.

Sometimes formfeeds in the source get passed to the assembler, causing it to crash.

There is no macro processor.

GRIPES

Complaints should be sent to:
Per Bothner (mail to bothner@score)

There is also a pc68 mailing list. To add yourself to it, send a message to mailer@su-whitney. The first line of the message body should say:
add me to pc68

To say messages to to list, mail to pc68 at shasta, diablo, navajo or whitney.

NAME

rmaker – resource maker (compiler) for Macintosh

SYNOPSIS

rmaker file.rc
rmaker [**–d** type] file.rc

DESCRIPTION

Rmaker reads an ascii resource compiler input file "file.rc" and produces a Macintosh executable binary "file.rsrc". See the Inside Mac manual "Putting Together a Macintosh Application" for a description of this format and process. It is also helpful to look at one of the example '.rc' files in the SUMacC 'mac/' source directory.

Most of the commonly used resource types are implemented: STR, HEXA, CODE, DRVR, ALRT, DITL, DLOG, WIND, MENU, CNTL, ICON, CURS, PAT, INIT and PACK. See the BUGS section below for exceptions.

The optional –d (debug) switch will list out in hex the contents of all resources matching that four letter type.

SEE ALSO

"Putting Together a Macintosh Application"

BUGS

If you have more than one resource of the same type, they must all be grouped together in the file, and each resource must begin with the "Type" statement.

Types NOT implemented: ICN#, PAT#, STR#, FREF, BNDL, FONT, GNRL. You can always use an "inherited type" of HEXA (e.g. Type BNDL = HEXA) to simulate unimplemented types. GNRL would be even better for this (if someone would only implement it!)

If you get the message "impossible relocation", it usually means your b.out had some undefined external references; check the error output from 'ld', you probably misspelled some global or routine name.

SUN RPC
REMOTE PROCEDURE CALL PACKAGE

Sun Microsystems

NAME

portmap – DARPA port to RPC program number mapper

SYNOPSIS

/usr/etc/rpc.portmap

DESCRIPTION

Portmap is a server that converts RPC program numbers into DARPA protocol port numbers. It must be running in order to make RPC calls.

When an RPC server is started, it will tell *portmap* what port number it is listening to, and what RPC program numbers it is prepared to serve. When a client wishes to make an RPC call to a given program number, it will first contact *portmap* on the server machine to determine the port number where RPC packets should be sent.

Normally, standard RPC servers are started by *inetd*(8c), so *portmap* must be started before *inetd* is invoked.

SEE ALSO

servers(5), rpcinfo(8), inetd(8)

BUGS

If *portmap* crashes, all servers must be restarted.

NAME

rpcinfo – report RPC information

SYNOPSIS

rpcinfo –p [host]
rpcinfo –u host program-number version-number
rpcinfo –t host program-number version-number

DESCRIPTION

Rpcinfo makes an RPC call to an RPC server and reports what it finds.

OPTIONS

–p Probe the portmapper on *host*, and print a list of all registered RPC programs. If *host* is not specified, it defaults to the value returned by *hostname*(1).

–u Make an RPC call to procedure 0 of *program-number* using UDP, and report whether a response was received.

–t Make an RPC call to procedure 0 of *program-number* using TCP, and report whether a response was received.

SEE ALSO

RPC Reference Manual, portmap(8)

X WINDOW SYSTEM
M.I.T.

NAME

X - A network transparent window system for Unix

DESCRIPTION

X is a network transparent windowing system developed at MIT which runs under Ultrix-32 Version 1.2 and 4.3BSD Unix.

X display servers run on computers with bitmap terminals. The server distributes user input to, and accepts output requests from various client programs located either on the same machine or elsewhere in the Internet. While a client normally runs on the same machine as the X server it is talking to, this need not be the case.

X supports overlapping windows, fully recursive subwindows, text and graphics operations within windows. For a full explanation of functions, see "Xlib - C Language X Interface" document.

When you first log in on a display running X, you are using the *xterm(1)* terminal emulator program. You need not learn anything extra to use a display running X as a terminal beyond moving the mouse cursor into the login window to log in normally.

X attempts to provide hooks for your favorite style of user interface; feel free to write your own if you don't like the style provided by existing window managers (see *xwm(1)*, *xnwm(1)*, or *uwm(1)*). These programs are used to manipulate existing top level windows, including moving, resizing, and iconifying existing windows. You should start your favorite window manager when you log in on a display running X.

Current client programs of X include a terminal emulator (*xterm(1)*), window managers (*xwm(1)*, *xnwm(1)* and *uwm(1)*), bitmap editor (*bitmap(1)*), access control program (*xhost(1)*), user preference setting program (*xset(1)*), load monitor (*xload(1)*), clock (*xclock(1)*), impress previewer (*ximpv(1)*), font displayer (*xfd(1)*), demos (*xdemo(1)*), and editors (e.g., *xted*). On some systems, mail notification has been integrated (*biff(1)*).

OPTIONS

The following options can be given on the command line to the X server, usually started by *init(1)* using information stored in the file */etc/ttys*. (see *ttys(5)*, *X(8c)* for details):

-a #	sets mouse acceleration (pixels)
-c	turns off key-click
c #	sets key-click volume (0-8)
-f #	sets feep(bell) volume (0-7)
-l	sets LockUpDownMode
l	sets LockToggleMode
m	forces "monochrome" mode on a color display
-p #	sets screen-saver pattern cycle time (minutes)
-r	turns off auto-repeat
r	turns on auto-repeat
-s #	sets screen-saver timeout (minutes)
-t #	sets mouse threshold (pixels)
v	sets video-on screen-saver preference
-v	sets video-off screen-saver preference
-0 *color*	sets color map entry 0 (BlackPixel)
-1 *color*	sets color map entry 1 (WhitePixel)
-D *rgbdb*	sets RGB database file

The defaults are "-a 4 c 6 -f 3 l -p 60 r -s 10 -t 2 -0 #008 -1 #ffffff -D /usr/lib/rgb".

X DEFAULTS

Many X programs follow the convention of using a file called *.Xdefaults* in your home directory to allow tailoring the default values of many items on the display (default font, border width, icon behavior, and so on). The format of this file is "programname.keyword:value", where the default value for each keyword is set to the specified string. If the program name is missing, the default "keyword" value is set to the value for all programs. Case is not significant in keywords. Any whitespace before the value is ignored. Any global defaults should precede program defaults in the file. See the manual pages for a list of what defaults can be set in a given program. Here is an overblown example *~/.Xdefaults* file.

```
0is a comment
.BorderWidth:                 2
.BitmapIcon:                  on
.MakeWindow.Background:       #8e8
.MakeWindow.Border:           #f26
.MakeWindow.BodyFont:         cor
.MakeWindow.Foreground:       medium slate blue
.MakeWindow.Freeze:           on
.MakeWindow.Mouse:            #e6f
.MakeWindow.MouseMask:        black
.MakeWindow.ClipToScreen:     on
.Menufreeze:                  on
.Menubackground:              maroon
.Panefont:                    8x13
.SelectionFont:               8x13
.SelectionBorder:             black
.Paneborderwidth:             1
xterm.Panespread:             .25
biff.Background:              violet red
biff.BodyFont:                9x15
biff.Border:                  black
biff.Foreground:              green yellow
biff.Mouse:                   coral
bitmap.Background:            forest green
bitmap.Border:                salmon
bitmap.Foreground:            white
bitmap.Highlight:             red
bitmap.Mouse:                 black
xclock.Background:            plum
xclock.Border:                black
xclock.Foreground:            red
xclock.Highlight:             blue
xclock.Mode:                  analog
xshell.action.LeftButton:     xterm =80x65-0+0 -fn 6x10
xshell.action.MiddleButton:   xted =80x65+0-0
xshell.action.RightButton:    xterm =20x20-0-0 -fn 6x10 -e dc
xshell.action.$:              xterm =80x65+0+0 -fn 6x10 -e sh
xshell.action.#:              xterm =80x65+0+0 -fn 6x10 -e su
xshell.ReverseVideo:          on
xshell.WindowGeometry:        =-0-0
xshell.Quiet:                 on
xdemo.Background:             white
xdemo.Border:                 black
xdemo.balls.Background:       maroon
xdemo.balls.Foreground:        white
xdemo.circles.Foreground:      khaki
xdemo.draw.Background:         light gray
xdemo.draw.BodyFont:           oldeng
xdemo.draw.Foreground:         midnight blue
xdemo.draw.Mouse:              white
xdemo.menulife.Background:     medium turquoise
xdemo.menulife.Foreground:     orange red
xdemo.menulife.MenuBackground:light blue
xdemo.menulife.MenuFont:       oldeng
xdemo.menulife.MenuForeground:dark orchid
xdemo.menulife.MenuMouse:      orange
xdemo.menulife.Mouse:          salmon
xdemo.plaid.Foreground:        red
xdemo.qix.Foreground:          violet red
xdemo.slide.Foreground:        forest green
xdemo.wallpaper.Foreground:    medium turquoise
xdemo.xor.Foreground:          blue violet
ximpv.Background:              dark green
ximpv.Border:                  red
ximpv.Foreground:              cyan
ximpv.Mouse:                   white
xload.Background:              #ff0068
xload.Border:                  black
xload.Foreground:              slate blue
xload.Highlight:               yellow
xload.ReverseVideo:            on
xted.Background:               firebrick
xted.BodyFont:                 kiltercrn
xted.Border:                   tan
xted.Cursor:                   yellow
xted.Foreground:               white
xted.Highlight:                goldenrod
xted.Mouse:                    cyan
xterm.Background:              #355
xterm.BodyFont:                6x13p
xterm.Cursor:                  green
xterm.Foreground:              white
xfax.Background:               white
xfax.Border:                   green
xfax.Foreground:               red
xfax.Mouse:                    blue
```

By default when you log in, only programs running on your local computer will be allowed to interact with your display. If someone else on a different machine wants to show you something, you can use the *xhost(1)* program to allow access to your display.

SIZING WINDOWS

Many programs ask you to manually size their top-level window. When started, such a program will typically popup an identification window in the upper left corner of the display. The window can be created with the center button: press the button to define one corner of the window, move the cursor to where the opposite corner of the window should be and release the button. For text applications, the left and right buttons can also be used. Pressing the left button typically produces an 80 by 24 window, which can then be moved around, and placed by releasing the button. Similarly, the right button typically produces an 80 by full screen window. For graphics applications, the left button typically creates a default size window in a default location, while the right button creates a default size window at the position of the cursor.

Most applications (e.g., *xted*, *xdemo*, and *xfax*) read options to control sizing of initial windows. The "MakeWindow.BodyFont" option controls the font for the popup window. The "MakeWindow.BorderWidth" and "MakeWindow.InternalBorder" options control the outer and inner borders. The "MakeWindow.ReverseVideo" option can be set to "on" to reverse colors. On color displays, the "MakeWindow.Foreground", "MakeWindow.Background", and "MakeWindow.Border" options control the color of the popup window, and the "MakeWindow.Mouse" and "MakeWindow.MouseMask" options control the color of the mouse cursor. The "MakeWindow.Freeze" option, when set to "on", will stop all other output while the window is sized, and use a steady outline instead of continuously flashing the window outline. The "MakeWindow.ClipToScreen" option will clip the resulting window to fit on the screen. (Currently only implemented in programs using the *XCreateTerm* subroutine.)

GEOMETRY SPECIFICATION

Most programs accept a geometry specification. This allows automatic creation and placement of windows on the screen at login and other convenient times. **=[WIDTH][xHEIGHT][{ -}XOFF[{ -}YOFF]]** The []'s denote optional parameters, the {}'s surround alternatives. WIDTH and HEIGHT are in number of characters for text oriented applications, and usually in pixels for graphics oriented applications. XOFF and YOFF are in pixels. If you don't give XOFF and/or YOFF, then you must use the mouse to create the window. If you give XOFF and/or YOFF, then a WIDTHxHEIGHT window will automatically be creating without intervention. XOFF and YOFF specify deltas from a corner of the screen to the corresponding corner of the window, as follows:

XOFF+YOFF	upper left to upper left
-XOFF+YOFF	upper right to upper right
XOFF-YOFF	lower left to lower left
-XOFF-YOFF	lower right to lower right

KEYBOARD

If you don't like the standard keyboard layout or the default definitions of keymap and function keys, the keyboards on most displays can be remapped to suit your taste. Many programs look for a file called *.Xkeymap* in your home directory. This is a binary file, produced from a source map with the *keycomp(1)* program.

COLORS

Many programs allow you to specify colors for background, border, text, etc. A color specification can be given either as an english name (see */usr/lib/rgb.txt* for defined names), or three hexadecimal values for the red, green, and blue components, in one of the following formats:

```
#RGB
#RRGGBB
#RRRGGGBBB
#RRRRGGGGBBBB
```

DISPLAY SPECIFICATION

When you first log in, the environment variable "DISPLAY" will be set to a string "machine:display" (for example, "mit-athena:0") which will determine which display an X application will talk to by default.

Most applications will also interpret an argument with a ":" in it to be the display to use.

When using DECnet, the format "node::display" should be used.

MENU DEFAULTS

As there is now a standard menu package for X (*XMenu(3x)*), you can tune the behavior of menus in programs using this package with a set of *Xdefaults*. *Xterm*'s 'Mode Menu' is controlled by these defaults for example.

MenuFreeze
Determines whether or not to grab the *X* server while a menu is posted. One of: on, off. The default value is off.

MenuStyle
Determines the menu display style. One of: left_hand, right_hand, center. The default value is right_hand.

MenuMode
Determines the menu selection high light mode. One of: box, invert. If box mode is chosen then the SelectionBorderWidth and SelectionBorderColor parameters effect the box line width and color respectively. If invert mode is chose then the SelectionForeground and MenuBackground colors are used for the inversion. The default value is invert.

MenuMouse
Determines the color of the mouse cursor while it is within the menu. Any valid *X* color may be used. The default value is black.

MenuBackground
Determines the menu background color. Any valid *X* color may be used. The default value is white.

MenuInactivePattern
Determines which of the five possible bitmap patterns will be used to tile inactive panes. One of: dimple1, dimple3, gray1, gray3, cross_weave. The default value is gray3.

PaneStyle
Determines the display style of all menu panes. One of: flush_left, flush_right, center. The default value is center.

PaneFont
Determines the font used for the label (heading text) of each pane. Any valid *X* font may be used. The default value is 8x13.

PaneForeground
Determines the pane foreground color. This is the color used for the label (heading text) in each pane. Any valid *X* color may be used. The default value is black.

PaneBorder
Determines the color of all menu pane borders. Any valid *X* color may be used. The default value is black.

PaneBorderWidth
Determines the width (in pixels) of all menu pane borders. Any integer greater than or equal to 0 may be used. The default value is 2.

PaneSpread
Determines the horizontal spread of menu panes. Any double greater than or equal to 0.0 may be used. A value of 1.0 specifies a one to one ratio between horizontal spread and vertical spread. A value less than 1.0 will compress the menu panes inward and a value greater than 1.0 will expand them outward. The default value is 1.0.

SelectionStyle
Determines the display style of all menu selections. One of: flush_left, flush_right, center. The default value is flush_left.

SelectionFont
Determines the font used for the text in each selection. Any valid X font may be used. The default value is 6x10.

SelectionForeground
Determines the selection foreground color. This is the color used for the text in each selection. Any valid *X* color may be used. The default value is black.

SelectionBorder
Determines the color of all menu selection borders. Any valid *X* color may be used. The default value is black.

SelectionBorderWidth
Determines the width (in pixels) of all menu selection borders. Any integer greater than or equal to 0 may be used. The default value is 1.

SelectionSpread
Determines the inter-selection spread. Any double greater than or equal to 0.0 may be used. A value of 1.0 specifies that 1.0 times the height of the current selection font will be used for padding The default value is 0.25.

SEE ALSO

X(8c), xterm(1), bitmap(1), xwm(1), xnwm(1), xhost(1), xclock(1), xload(1), xset(1), keycomp(1), xdemo(1), biff(1), qv(4), vs(4), init(8), ttys(5), uwm(1), xrefresh(1), xwininfo(1), ximpv(1), xdvi(1), pikapix(1), xwd(1), xwud(1), xinit(1), xted(1), xdemo(1), Xqvss(8c), Xvs100(8c), Xsun(8c), Xnest(8c)
'Xlib - C Language X Interface'

AUTHORS

It is no longer feasible to list all people who have contributed something to X; below is a short list of people who have added significant code to device independent parts of X.
Bob Scheifler (MIT-LCS), Jim Gettys (MIT-Project Athena, DEC), Mark Vandevoorde (MIT-Project Athena, DEC), Tony Della Fera (MIT-Project Athena, DEC), Ron Newman (MIT-Project Athena, MIT), Shane Hartman and Stuart Malone (MIT-LCS), Doug Mink (Smithsonian Astrophysical Observatory), Bob McNamara (DEC-MAD), and Stephen Sutphen (University of Alberta).

Special thanks must go to Paul Asente (of DECWRL and Stanford University), who wrote "W" which saved us much time and energy early in this project, and who is now an active X contributor as well, and Chris Kent (of DECWRL and Purdue University) who both struggled mightily (and won!) to turn the Vs100 into something useful under Unix.

We are very grateful for the interest shown by many groups in the country, which has encouraged us to make X more than our personal toy. Great thanks must go to Digital's Ultrix Engineering Group for the QDSS implementation, and to Digital's Workstations Group for the QVSS implementation.

NAME

XMenu - X Deck of cards Menu System

SYNOPSIS

```
#include <X/XMenu.h>

XMenu *XMenuCreate(parent, xdef_env)
Window parent;
char *xdef_env;

int XMenuAddPane(menu, label, active)
XMenu *menu;
char *label;
int active;

int XMenuAddSelection(menu, pane, data, label, active)
XMenu *menu;
int pane;
char *data;
char *label;
int active;

int XMenuInsertPane(menu, pane, label, active)
XMenu *menu;
int pane;
char *label;
int active;

int XMenuInsertSelection(menu, pane,selection, data, label, active)
XMenu *menu;
int pane, selection;
caddr_d data;
char *label;
int active;

int XMenuFindPane(menu, label)
XMenu *menu;
char *label;

int XMenuFindSelection(menu, pane, label)
XMenu *menu;
int pane;
char *label;

int XMenuChangePane(menu, pane, label)
XMenu *menu;
int pane;
char *label;

int XMenuChangeSelection(menu, pane,selection, data,d_sw, label,l_sw)
XMenu *menu;
int pane, selection;
char *data;
int d_sw;
char *label;
int l_sw;

int XMenuSetPane(menu, pane, active)
XMenu *menu;
int pane;
int active;

int XMenuSetSelection(menu, pane, selection, active)
XMenu *menu;
int pane, selection;
int active;

int XMenuDeletePane(menu, pane)
XMenu *menu;
int pane;

int XMenuDeleteSelection(menu, pane, selection)
XMenu menu;
int pane, selection;

int XMenuRecompute(menu)
XMenu *menu;

XMenuEventHandler(handler)
int (*handler)();

int XMenuLocate(menu, pane,selection, x,y, ulx,uly, width,height)
XMenu *menu;
int pane, selection;
int x, y;
int *ulx, *uly;
int *width, *height;

XMenuSetFreeze(menu, freeze)
XMenu *menu;
int freeze;

int XMenuActivate(menu, pane,selection, x,y, event_mask, data)
XMenu *menu;
int *pane, *selection;
int x, y;
int event_mask;
char **data;

XMenuDestroy(menu)
XMenu *menu;

char *XMenuError()
```

DESCRIPTION

XMenu is an *X* Window System Utility Package that implements a 'deck of cards' menu system. *XMenu* is intended for use in conjunction with *Xlib*, the *C Language X Window System Interface Library*.

In a 'deck of cards' menu system a menu is composed of several cards or panes. The panes are stacked as if they were a deck of playing cards that were fanned out. Each of these panes has one or more selections. A user interacts with a 'deck of cards' menu by sliding the mouse cursor across the panes of the menu. As the mouse cursor enters each pane it will rise to the top of the deck and become 'current'. If the current pane is an active pane it will be 'activated', or made available for selection. To indicate this its background will then change from the patterned inactive background to a solid color and the selections on that pane will be activated. If the current pane is not an active pane (a setable state) then it will not be activated. To indicate this its background will continue to be the patterned inactive background and no selections on the pane will be activated. The pane previously containing the mouse will lower (preserving its stacking order). If it was activated it will then become deactivated, its background changing back to the inactive pattern. Because of this action it is not possible to have more than one current pane at any one time. When the mouse cursor enters an active selection in a pane that has been activated then that selection will become activated and be high lighted. If the selection is not active or the pane has not been activated then the selection will not be activated and will not be high lighted. Selection high lighting is accomplished in one of two ways depending upon the state of the user's *Xdefaults* variables. If 'box' mode high lighting is in effect, the menu selection will be activated by placing a high light box around the selection as the mouse cursor enters the selection's active region and removing it (deactivating the selection) as the cursor leaves. If 'invert' mode high lighting is in effect, the menu selection will be activated by inverting the background and foreground colors within the selection's active region as the mouse cursor enters it and reinverting them as the cursor leaves.

The application specifies a mouse event that will signify that the user has made a selection. Any time that the selection mouse event is received by *XMenu* one of several results will occur, depending upon the state of the menu system at the time of the event. If the selection event occurs while the mouse cursor is in an activated selection the data that has been stored with that selection will be returned to the application program. The data stored is in the form of a generic pointer to memory (char *). This allows the application programmer to completely define the interpretation of the selection data by recasting the data pointer as is desired.

An application constructs a menu by first creating the *XMenu* object. Once the *XMenu* object has been created then panes and selections are added in order as is needed. Typically panes contain related selections that are 'described' by the pane's label. For example, you might create a pane labeled 'Mail' that has selections labeled 'Read', 'Send', 'Forward', 'Refile' and 'Delete'. There is no real need for the panes in a menu to be related to each other but typically they are related by default by the fact that they are all being utilized the application that created the menu.

The *XMenu* system is maintained (menus, panes and selections) via routines in the *XMenu* library. The library contains the following routines:

XMenuCreate

In order for a process to create a menu, it is necessary for that process to have opened a connection to an *X* display server and have a window in hand that will be designated as the parent window of the menu being created (remember that *X* is designed such that child windows of a parent window are clipped to the borders of the parent). Typically the *X* root window (*RootWindow*) is used for this purpose. When the connection is open and a parent window chosen, the application calls *XMenuCreate* passing it the parent window and a null-terminated string. The string designates the default environment name that will be used by XMenu to read the users *Xdefaults* variables. Typically the application name is used for this purpose (a good software engineering practice is to use element zero of the applications argument vector, argv[0], as the default environment since this is the name by which the application was called from the shell). All *user* setable parameters are set via the *Xdefaults* mechanism. If any parameters do not have *Xdefaults* values then they default to preset *XMenu* internal values. The *Xdefaults* parameters are listed below along with their preset internal values. If the create operation is successful *XMenuCreate* will return an *XMenu* object. If it fails NULL will be returned.

XMenuAddPane

Once a menu has been created the application may then begin adding panes and subsequently selections. Panes are added by calling *XMenuAddPane. XMenuAddPane* adds additional panes to a menu in call order. That is, panes will appear in the menu with the first pane added being at the front of the pane stack and the last pane added being at the back of the pane stack. *XMenuAddPane* takes the following arguments: The menu to which the pane is being added; A null-terminated string that will be the label for the new pane; and an flag that designates whether or not the pane is to be considered active for selection. It is sometimes useful to add inactive panes to indicate a currently unavailable but planned set of selections. If the add operation is successful the index number of the pane just added will be returned. If it fails XM_FAILURE will be returned. Further panes may be added at a later time but remember that when this routine is used to add panes they are always added to the back of the pane stack!

XMenuAddSelection

Once a pane has been added to a menu is it possible to begin adding selections to that pane. Selections are added to panes in much the same way as panes are added to menus. Selections are added by calling *XMenuAddSelection. XMenuAddSelection* adds additional selections to a pane in call order. That is, selections will appear in the pane with the first selection added being at the top of the pane and the last selection added being at the bottom of the pane. *XMenuAddSelection* takes the following arguments: The menu containing the pane to which the selection is being added; The index number of the pane to which the selection is being added; A null-terminated string that will be the label for the new selection; A (char *) data value that will be returned by *XMenuActivate* whenever the new selection is selected by the menu's user; and a flag that designates whether or not the selection will be considered active. It is sometimes useful to add inactive selections which may become active as the application state changes. If the add operation is successful then the index number of the selection just added will be returned. If it fails XM_FAILURE will be returned. Further selections may be added at a later time but remember when this routine is used to add selections they are always added to the bottom of a pane!

XMenuInsertPane

This routine allows the application to insert menu panes into a menu in random order. If the index number of the pane being inserted matches the index number of a pane that already exists, then the existing pane is displaced backward (its index number and the index numbers of all following planes increased by one) in the menu and the new pane inserted in its place. Panes may be inserted into any menu provided that the index number of the pane being inserted is no more than one greater than the index number of the last pane in the menu. For example, if a menu contains 4 panes with index numbers 0 through 3 then it is possible to insert a new pane with an index number from 0 through 4 inclusive. It is possible to use *XMenuInsert-Pane* in place of *XMenuAddPane* but in situations where panes are simply being added to a menu one after another then the use of the simpler and more efficient *XMenuAddPane* routine is encouraged. *XMenuInsertPane* takes the following arguments: The menu into which the pane is being inserted; the index number of the new pane; a null-terminated string that will be the label for the new pane; and an int that designates whether or not the pane will to be considered active for selection. It is sometimes useful to add inactive panes to indicate a currently unavailable but planned set of selections. If the insert operation is successful the index number of the pane just inserted will be returned. If it fails XM_FAILURE will be returned.

XMenuInsertSelection

This routine allows the application to insert selections into a menu pane in random order. If the index number of the selection being inserted matches the index number of a selection that already exists in the specified pane, then the existing selection is displaced downward (its index number and the index numbers of all following selections increased by one) in the pane and the new selection inserted in its place. Selections may be inserted into any pane provided that the index number of the selection being inserted is no more than one greater than the index number of the last selection in the pane. For example, if a pane contains 4 selections numbered 0 through 3 then it is possible to insert a new selection with an index number from 0 through 4 inclusive. It is possible to use *XMenuInsertSelection* in place of *XMenuAddSelection* but in situations where selections are simply being added to a pane one after another then the use of the simpler and more efficient *XMenuAddSelection* routine is encouraged. *XMenuInsertSelection* takes the following arguments: the menu containing the pane into which the selection is being inserted; the index number of the pane to which the selection is being inserted; the desired index number of the new selection; a null-terminated string that will be the label for the new selection; A (char *) data value that will be returned by *XMenuActivate* whenever the new selection is selected by a user; and an int that designates whether or not the selection will be considered active for selection. It is sometimes useful to insert inactive selections which may become active as the application state changes. If the insert operation is successful the index number of the selection just inserted will be returned. If it fails XM_FAILURE will be returned.

XMenuFindPane

This routine allows the application to find the index number of a pane whose label matches a given NULL terminated string. *XMenuFindPane* takes the following arguments: the menu containing the pane whose index number is being searched for; and a null terminated string to be searched for. If the find operation is successful then the index number of the first pane whose label matches the given string will be returned. If it fails XM_FAILURE will be returned.

XMenuFindSelection

This routine allows the application to find the index number of a selection whose label matches a given NULL terminated string. *XMenuFindSelection* takes the following arguments: the menu containing the pane which contains the selection being searched for; the index number of the pane which contains the selection being searched for; and a null terminated string to be searched for. If the find operation is successful then the index number of the first selection whose label matched the given string will be returned. If is fails XM_FAILURE will be returned.

XMenuChangePane

This routine allows the application to change a pane's label on the fly. This is useful for situations where a state change in the application must be reflected in the menu. *XMenuChangePane* takes the following arguments: the menu containing the pane whose label is being changed; the index number of that pane in the specified menu; and a null-terminated string that will be the used as the new pane label. If the change operation is successful the index number of the pane just changed will be returned. If it fails XM_FAILURE will be returned. *XMenuChangePane* may be called any time after the pane being changed has been added / inserted into the specified menu.

XMenuChangeSelection

This routine allows the application to change a selection's data and label on the fly. This is useful for situations where a state change in the application must be reflected in the menu. *XMenuChangeSelection* takes the following arguments: the menu containing the pane that contains the selection to be changed; the index number of that pane in the menu; the index number of the selection to be changed; a (char *) new data value for the selection; an int that indicates whether or not to actually store the new data value (in case only the label is being changed); Aanull-terminated string that will be the used as the new selection label; and an int that indicates whether or not to actually store the new label (incase only the data value is being changed). If the change operation is successful the index number of the selection just changed will be returned. If it fails XM_FAILURE will be returned. *XMenuChangeSelection* may be called anytime after the pane selection being changed has been added to the specified pane and menu.

XMenuSetPane
XMenuSetPane allows the application to make an active pane inactive or an inactive pane active. This provides the application with the ability to restrict the usage of certain panes to times when they may or may not have a valid purpose. In addition this allows the application to activate and utilize dummy panes that were added at menu creation time as place holders for future selections. *XMenuSetPane* takes the following arguments: the menu containing the pane to be activated or deactivated; the index number of that pane in the specified menu; and an int that designates whether or not the pane is to be considered active for selection. If the set operation is successful the index number of the pane just set will be returned. If it fails XM_FAILURE will be returned. *XMenuSetPane* may be called anytime after the pane being set has been added / inserted into the specified menu.

XMenuSetSelection
XMenuSetSelection allows the application to make an active selection inactive or an inactive selection active. This provides the application with the ability to restrict the usage of certain selections to times when they may or may not have a valid purpose. In addition this allows the application to activate and utilize selections that were added at menu creation time with a future purpose in mind. *XMenuSetSelection* takes the following arguments: the menu containing the pane that contains the selection to be activated or deactivated; the index number of that pane in the menu; the index number of the selection to be activated / deactivated; and an int that designates whether or not to make the specified selection active. If the set operation is successful the index number of the selection just set will be returned. If it fails XM_FAILURE will be returned. *XMenuSetSelection* may be called anytime after the pane selection being set has been added to the specified pane and menu.

XMenuDeletePane
This routine allows the application to delete panes when they will no longer be needed. *XMenuDeletePane* takes the following arguments: the menu containing the pane to be deleted; and the index number of that pane in the specified menu.

XMenuDeleteSelection
This routine allows the application to delete selections when they will no longer be needed. *XMenuDeleteSelection* takes the following arguments: the menu containing the pane which contains the selection to be deleted; the index number of the pane containing the selection to be deleted; and the index number of the selection to be deleted in that pane.

XMenuRecompute
After the initial menu configuration has been constructed (in fact, anytime that the menu configuration, a pane label or selection label is altered), the menu dependencies need to be recomputed. *XMenu* will do this automatically if needed when *XMenuLocate* or *XMenuActivate* is called. In the interest of efficiency it is suggested that the application call *XMenuRecompute* prior to any calls to *XMenuLocate* or *XMenuActivate.* This need only be done if *XMenuAddPane, XMenuAddSelection, XMenuInsertPane, XMenuInsertSelection, XMenuChangePane, XMenuChangeSelection, XMenuDeletePane,* or *XMenuDeleteSelection* have been called since the last call to *XMenuRecompute* or *XMenuActivate.* If *XMenuRecompute* is called before the first pane has been added to the menu a error will result indicating that the menu has not been initialized. The most efficient state is achieved if a sequence of panes and selections are added or modified in order and then a single call is immediately made to *XMenuRecompute.* In this way all operations will batched and all dependencies will

be up to date by the time the next *XMenuActivate* call occurs. If the recompute operation is successful XM_SUCCESS will be be returned. If it fails XM_FAILURE will be returned.

XMenuEventHandler

Since *XMenu* shares the *Xlib* event queue with the application, it is possible that *X* events selected by the application will arrive and be queued while a menu is posted. Before a menu is posted, it is up to the application to decide what will happen to events that do occur while the menu is posted. *XMenuEventHandler* allows the application to specify an asynchronous event handling routine. *XMenuEventHandler* takes only one argument which is a pointer to a routine which returns int. This routine will be called by *XMenuActivate* if it encounters an event that it does not recognize. The format of the handler should be as follows:
int handler(event)
XEvent *event;
If no action is taken by the application (i.e., no event handler is specified) *XMenuActivate* will discard any events that they do not recognize.

XMenuLocate

This routine provides an application will all the necessary data to properly locate and position a menu with respect to the parent window. *XMenuLocate* takes the following arguments: the menu that is being located; the index number of the current pane; the index number of the current selection; the X and Y coordinates of where the application would like the center of the current selection (in the current pane) to be; and four return value pointers to int that will be filled in by the routine. The four return value pointers are set to the following values (respectively): the upper left X and Y coordinates of the entire menu (relative to the parent window); and the overall width and height of the entire menu. If the locate operation is successful XM_SUCCESS will be be returned. If it fails XM_FAILURE will be returned.

XMenuSetFreeze

This routine allows the application to forcibly override the *Xdefaults* setting of the 'freeze' parameter. If freeze mode is turned on the bits under where the menu will appear are saved by *XMenu* then the *X* server is frozen and remains frozed while the menu is activated. Immediately after the menu is deactivated the bits under the menu are restored to their original state and the server is unfrozen. This routine is necessary for certain applications that must guarantee that the screen contents are not damaged by *XMenu. XMenuSetFreeze* takes two arguments: The menu to be set and an int that indicates whether or not to place the menu in freeze mode.

XMenuActivate

XMenuActivate maps a given menu to user selection. Before *XMenuActivate* is called it is suggested that the application synchronize the X connection and and process all events in the *Xlib* internal event queue. This guarantees that a minimum of asynchronous call-backs to the applications event handler routine (or discards if no application event handler is specified). *XMenuActivate* guarantees that no unprocessed events of its own will be left in the *Xlib* event queue upon its return. *XMenuActivate* takes the following arguments: the menu that is to be posted; the desired current pane and selection; the X and Y menu position; the mouse button event mask; and a pointer to a pointer to char (char **). The menu is positioned within the menu's parent window such that the specified X and Y location (relative to the parent window) is in the center of the specified current selection in the current pane. The mouse button event mask provided by the application should be suitable for an *XGrabMouse* operation. It provides the application with a way to indicate which mouse events will be used to identify a selection request. Every time *XMenuActivate* returns, the pane and selection indices are left at their last known values (i.e., the last current pane and selection indices). The following are the defined return states for this routine:

1) If the selection that is current at the time a selection request is made is active then the data pointer will be set to the data associated with that particular selection and XM_SUCCESS is returned.
2) If the selection that is current at the time a selection request is made is not active then the data pointer will be left untouched and XM_IA_SELECT will be returned.
3) If there is no selection current at the time a selection request is made then the data pointer will be left untouched and XM_NO_SELECT will be returned.
4) If at any time an error occurs the data pointer is left untouched and XM_FAILURE is returned.

XMenuDestroy

When the application is no longer intending to use a menu *XMenuDestroy* should be called. *XMenuDestroy* frees all resources (both *X* resources and system resources) that are being held by the menu. *XMenuDestroy* takes only one argument, the menu to be destroyed. WARNING! Using a menu after it has been destroyed is to invite disaster!

XMenuError

When called *XMenuError* will return a null-terminated string that describes the current error state of the *XMenu* library. The string returned is static in the *XMenu* library and should not be modified or freed. The error state is set every time an *XMenu* routine returns a status condition. *XMenuError* takes no arguments.

X DEFAULTS

MenuFreeze

Determines whether or not to grab the *X* server while a menu is posted. One of: on, off. The default value is off.

MenuStyle

Determines the menu display style. One of: left_hand, right_hand, center. The default value is right_hand.

MenuMode

Determines the menu selection high light mode. One of: box, invert. If box mode is chosen then the SelectionBorderWidth and SelectionBorderColor parameters effect the box line width and color respectively. If invert mode is chose then the Selection-Foreground and MenuBackground colors are used for the inversion. The default value is invert.

MenuMouse

Determines the color of the mouse cursor while it is within the menu. Any valid *X* color may be used. The default value is black.

MenuBackground

Determines the menu background color. Any valid *X* color may be used. The default value is white.

MenuInactivePattern

Determines which of the five possible bitmap patterns will be used to tile inactive panes. One of: dimple1, dimple3, gray1, gray3, cross_weave. The default value is gray3.

PaneStyle
Determines the display style of all menu panes. One of: flush_left, flush_right, center. The default value is center.

PaneFont
Determines the font used for the label (heading text) of each pane. Any valid *X* font may be used. The default value is 8x13.

PaneForeground
Determines the pane foreground color. This is the color used for the label (heading text) in each pane. Any valid *X* color may be used. The default value is black.

PaneBorder
Determines the color of all menu pane borders. Any valid *X* color may be used. The default value is black.

PaneBorderWidth
Determines the width (in pixels) of all menu pane borders. Any integer greater than or equal to 0 may be used. The default value is 2.

PaneSpread
Determines the horizontal spread of menu panes. Any double greater than or equal to 0.0 may be used. A value of 1.0 specifies a one to one ratio between horizontal spread and vertical spread. A value less than 1.0 will compress the menu panes inward and a value greater than 1.0 will expand them outward. The default value is 1.0.

SelectionStyle
Determines the display style of all menu selections. One of: flush_left, flush_right, center. The default value is flush_left.

SelectionFont
Determines the font used for the text in each selection. Any valid X font may be used. The default value is 6x10.

SelectionForeground
Determines the selection foreground color. This is the color used for the text in each selection. Any valid *X* color may be used. The default value is black.

SelectionBorder
Determines the color of all menu selection borders. Any valid *X* color may be used. The default value is black.

SelectionBorderWidth
Determines the width (in pixels) of all menu selection borders. Any integer greater than or equal to 0 may be used. The default value is 1.

SelectionSpread
Determines the inter-selection spread. Any double greater than or equal to 0.0 may be used. A value of 1.0 specifies that 1.0 times the height of the current selection font will be used for padding The default value is 0.25.

DIAGNOSTICS

Since *XMenu* uses the *Xlib* library, the *XIOError* and *XError Xlib* routines may be set by the application to change how asynchronous error reporting occurs.

Synchronous error reporting is primarily accomplished by examining the return values of routines and using the *XMenuError* routine. Although its use is discouraged, synchronous error reporting may also be accomplished by having the application directly examine the value of the *_XMErrorCode* global variable. *_XMErrorCode* is set every time an *XMenu* routine returns a status condition. The following sequence of symbols is provided in *XMenu.h* and may be used to index the null-terminated description strings in the global error string array *_XMErrorList.*

XME_CODE_COUNT	Total number of entries in *_XMErrorList* (17).
XME_NO_ERROR	-> "No error"
XME_NOT_INIT	-> "Menu not initialized"
XME_ARG_BOUNDS	-> "Argument out of bounds"
XME_P_NOT_FOUND	-> "Pane not found"
XME_S_NOT_FOUND	-> "Selection not found"
XME_STYLE_PARAM	-> "Invalid menu style parameter"
XME_GRAB_MOUSE	-> "Unable to grab mouse"
XME_INTERP_LOC	-> "Unable to interpret locator"
XME_CALLOC	-> "Unable to calloc memory"
XME_CREATE_ASSOC	-> "Unable to create XAssocTable"
XME_STORE_BITMAP	-> "Unable to store bitmap"
XME_MAKE_TILES	-> "Unable to make tile pixmaps"
XME_MAKE_PIXMAP	-> "Unable to make pixmap"
XME_CREATE_CURSOR	-> "Unable to create cursor"
XME_OPEN_FONT	-> "Unable to open font"
XME_CREATE_WINDOW	-> "Unable to create windows"
XME_CREATE_TRANSP	-> "Unable to create transparencies"

FILES

/usr/include/X/XMenu.h, /usr/lib/libXMenu.a, /usr/include/X/Xlib.h, /usr/lib/libX.a

SEE ALSO

Xlib(3x), X(1), X(8c)

AUTHOR

Copyright 1985, 1986, Massachusetts Institute of Technology.

See *X(1)* for a complete copyright notice.

Tony Della Fera (MIT Project Athena, DEC)

BUGS

There is a problem that necessitates an additional round trip time when panes are activated and deactivated. In order for this to be fixed efficiently, a change needs to be made to the *X* protocol.

NAME

uwm - Window Manager Client Application of X

SYNTAX

uwm [-f *filename*]

DESCRIPTION

The *uwm* command is a window manager client application of the window server.

When the command is invoked, it traces a predefined search path to locate any *uwm* startup files. If no startup files exist, *uwm* initializes its built-in default file.

If startup files exist in any of the following locations, it adds the variables to the default variables. In the case of contention, the variables in the last file found override previous specifications. Files in the *uwm* search path are:

/usr/new/lib/X/uwm/system.uwmrc
$HOME/.uwmrc

To use only the settings defined in a single startup file, include the variables, **resetbindings**, **resetmenus**, **resetvariables** at the top of that specific startup file.

ARGUMENTS

-f *filename*
Names an alternate file as a *uwm* startup file.

STARTUP FILE VARIABLES

Variables are typically entered first, at the top of the startup file. By convention, **resetbindings**, **resetmenus**, and **resetvariables** head the list.

autoselect/noautoselect
places menu cursor in first menu item. If unspecified, menu cursor is placed in the menu header when the menu is displayed.

delta=*pixels* indicates the number of pixels the cursor is moved before the action is interpreted by the window manager as a command. (Also refer to the **delta** mouse action.)

freeze/nofreeze locks all other client applications out of the server during certain window manager tasks, such as move and resize.

grid/nogrid displays a finely-ruled grid to help you position an icon or window during resize or move operations.

hiconpad=*n* indicates the number of pixels to pad an icon horizontally. The default is five pixels.

hmenupad=*n* indicates the amount of space in pixels, that each menu item is padded above and below the text.

iconfont=*fontname*
names the font that is displayed within icons. Font names are listed in the font directory, */usr/new/lib/X/font*.

maxcolors=*n* limits the number of colors the window manager can use in a given invocation. If set to zero, or not specified, *uwm* assumes no limit to the number of colors it can take from the color map. **maxcolors** counts colors as they are included in the file.

normali/nonormali
places icons created with **f.newiconify** within the root window, even if it is placed partially off the screen. With **nonormali** the icon is placed exactly where the cursor leaves it.

normalw/nonormalw
: places window created with **f.newiconify** within the root window, even if it is placed partially off the screen. With **nonormalw** the window is placed exactly where the cursor leaves it.

push=*n*
: moves a window *n* number of pixels or a relative amount of space, depending on whether **pushabsolute** or **pushrelative** is specified. Use this variable in conjunction with **f.pushup, f.pushdown, f.pushright,** or **f.pushleft.**

pushabsolute/pushrelative
: **pushabsolute** indicates that the number entered with push is equivalent to pixels. When an f.push (left, right, up, or down) function is called, the window is moved exactly that number of pixels.

  pushrelative indicates that the number entered with the push variable represents a relative number. When an f.push function is called, the window is invisibly divided into the number of parts you entered with the push variable, and the window is moved one part.

resetbindings, **resetmenus**, and **resetvariables**
: resets all previous function bindings, menus, and variables entries, specified in any startup file in the *uwm* search path, including those in the default environment. By convention, these variables are entered first in the startup file.

resizefont=*fontname*
: identifies the font of the indicator that displays in the corner of the window as you resize windows. See the */usr/new/lib/X/font* directory for a list of fonts.

reverse/noreverse
: defines the display as black characters on a white background for the window manager windows and icons.

viconpad=*n*
: indicates the number of pixels to pad an icon vertically. Default is five pixels.

vmenupad=*n*
: indicates the amount of space in pixels that the menu is padded on the right and left of the text.

volume=*n*
: increases or decreases the base level volume set by the *xset(1)* command. Enter an integer from 0 to 7, 7 being the loudest.

zap/nozap
: causes ghost lines to follow the window or icon from its previous default location to its new location during a move or resize operation.

BINDING SYNTAX

"*function*=[*control key(s)*]:[*context*]:*mouse events*:" *menu name* "

Function and mouse events are required input. Menu name is required with the *f.menu* function definition only.

Function

f.beep
: emits a beep from the keyboard. Loudness is determined by the volume variable.

f.circledown
: causes the top window that is obscuring another window to drop to the bottom of the stack of windows.

f.circleup
: exposes the lowest window that is obscured by other windows.

f.continue
: releases the window server display action after you stop action with the **f.pause** function.

f.focus	directs all keyboard input to the selected window. To reset the focus to all windows, invoke *f.focus* from the root window.
f.iconify	when implemented from a window, this function converts the window to its respective icon. When implemented from an icon, f.iconify converts the icon to its respective window.
f.lower	lowers a window that is obstructing a window below it.
f.menu	invokes a menu. Enclose 'menu name' in quotes if it contains blank characters or parentheses. **f.menu**=[*control key(s)*]:[*context*]:*mouse events*:" *menu name* "
f.move	moves a window or icon to a new location, which becomes the default location.
f.moveopaque	moves a window or icon to a new screen location. When using this function, the entire window or icon is moved to the new screen location. The grid effect is not used with this function.
f.newiconify	allows you to create a window or icon and then position the window or icon in a new default location on the screen.
f.pause	temporarily stops all display action. To release the screen and immediately update all windows, use the **f.continue** function.
f.pushdown	moves a window down. The distance of the push is determined by the push variables.
f.pushleft	moves a window to the left. The distance of the push is determined by the push variables.
f.pushright	moves a window to the right. The distance of the push is determined by the push variables.
f.pushup	moves a window up. The distance of the push is determined by the push variables.
f.raise	raises a window that is being obstructed by a window above it.
f.refresh	results in exposure events being sent to the window server clients for all unobscured or partially obscured windows. The windows will not refresh correctly if the exposure events are not handled properly.
f.resize	resizes an existing window. Note that some clients, notably editors, react unpredictably if you resize the window while the client is running.
f.restartn	causes the window manager application to restart, retracing the *uwm* search path and initializing the variables it finds.

Control Keys

By default, the window manager uses meta as its control key. It can also use ctrl, shift, lock, or null (no control key). Control keys must be entered in lower case, and can be abbreviated as: c, l, m, s for ctrl, lock, meta, and shift, respectively.

You can bind one, two, or no control keys to a function. Use the bar (|) character to combine control keys.

Note that client applications other than the window manager use the shift as a control key. If you bind the shift key to a window manager function, you can not use other client applications that require this key.

Context

The context refers to the screen location of the cursor when a command is initiated. When you include a context entry in a binding, the cursor must be in that context or the function will not be activated. The window manager recognizes the following four contexts: icon, window, root, (null).

The root context refers to the root, or background window, A (null) context is indicated when the context field is left blank, and allows a function to be invoked from any screen location. Combine contexts using the bar (|) character.

Mouse Buttons

Any of the following mouse buttons are accepted in lower case and can be abbreviated as l, m, or r, respectively: left, middle, right.

With the specific button, you must identify the action of that button. Mouse actions can be:

down	function occurs when the specified button is pressed down.
up	function occurs when the specified button is released.
delta	indicates that the mouse must be moved the number of pixels specified with the delta variable before the specified function is invoked. The mouse can be moved in any direction to satisfy the delta requirement.

MENU DEFINITION

After binding a set of function keys and a menu name to **f.menu**, you must define the menu to be invoked, using the following syntax:

```
        menu = " menu name " {
        "item name" : "action"
              .
              .
              .
        }
```

Enter the menu name exactly the way it is entered with the **f.menu** function or the window manager will not recognize the link. If the menu name contains blank strings, tabs or parentheses, it must be quoted here and in the f.menu function entry. You can enter as many menu items as your screen is long. You cannot scroll within menus.

Any menu entry that contains quotes, special characters, parentheses, tabs, or strings of blanks must be enclosed in double quotes. Follow the item name by a colon (:).

Menu Action

Window manager functions

Any function previously described. E.g., **f.move** or **f.iconify**.

Shell commands

Begin with an exclamation point (!) and set to run in background. You cannot include a new line character within a shell command.

Text strings

Text strings are placed in the window server's cut buffer.

Strings with a new line character must begin with an up arrow (^), which is stripped during the copy operation.

Strings without a new line must begin with the bar character (|), which is stripped during the copy operation.

Color Menus

Use the following syntax to add color to menus:

```
menu = "menu name" (color1:color2:color3:color4) {
"item name"  : (color5 :color6)  : " action "
        .
        .
        .
}
```

color1	Foreground color of the header.
color2	Background color of the header.
color3	Foreground color of the highlighter, the horizontal band of color that moves with the cursor within the menu.
color4	Background color of the highlighter.
color5	Foreground color for the individual menu item.
color6	Background color for the individual menu item.

Color Defaults

Colors default to the colors of the root window under any of the following conditions:

1) If you run out of color map entries, either before or during an invocation of *uwm*.

2) If you specify a foreground or background color that does not exist in the RGB color database (*/usr/lib/rgb.txt*) both the foreground and background colors default to the root window colors.

3) If you omit a foreground or background color, both the foreground and background colors default to the root window colors.

4) If the total number of colors specified in the startup file exceeds the number specified in the *maxcolors* variable.

5) If you specify no colors in the startup file.

EXAMPLES

The following sample startup file shows the default window manager options:

```
# Global variables
#
resetbindings;resetvariables;resetmenus
autoselect
delta=25
freeze
grid
hiconpad=5
hmenupad=6
iconfont=oldeng
menufont=timrom12b
resizefont=9x15
viconpad=5
vmenupad=3
volume=7
#
# Mouse button/key maps
```

```
#
# FUNCTION   KEYS  CONTEXT  BUTTON    MENU(if any)
# ========   ====  =======  ======    ============
f.menu =      meta  :    :left down   :"WINDOW OPS"
f.menu =      meta  :    :middle down :"EXTENDED WINDOW OPS"
f.move =      meta  :w|i  :right down
f.circleup =  meta  :root :right down
#
# Menu specifications
#
menu = "WINDOW OPS" {
"(De)Iconify":  f.iconify
Move:           f.move
Resize:         f.resize
Lower:          f.lower
Raise:          f.raise
}

menu = "EXTENDED WINDOW OPS" {
Create Window:                          !"xterm &"
Iconify at New Position:     f.lowericonify
Focus Keyboard on Window:    f.focus
Freeze All Windows:          f.pause
Unfreeze All Windows:        f.continue
Circulate Windows Up:        f.circleup
Circulate Windows Down:              f.circledown
}
```

RESTRICTIONS

The color specifications have no effect on a monochrome system.

FILES

/usr/lib/rgb.txt
/usr/new/lib/X/font
/usr/skel/.uwmrc
/usr/new/lib/X/uwm/system.uwmrc
$HOME/.uwmrc

SEE ALSO

X(1), X(8C)

AUTHOR

PLACED ON THE DERIVATIVE WORK IN ADDITION TO THAT SET FORTH ABOVE."

M. Gancarz, DEC Ultrix Engineering Group, Merrimack, New Hampshire, using some algorithms originally by Bob Scheifler, MIT Laboratory for Computer Science

NAME

xdvi – DVI Previewer for the X Window System

SYNOPSIS

xdvi [-s *shrink*] [-p *pixels*] [-l] [-rv] [-fg *color*] [-bg *color*] [-hl *color*] [-bd *color*] [-ms *color*] [=*geometry*] [*host:display*] file

DESCRIPTION

Xdvi is a program which runs under the X window system. It is used to preview DVI files, such as produced by TeX.

The **-p** option defines the pixels per inch for font selection. Default value is 300.

The **-s** option defines the initial shrink factor. Default value is 4.

The **-l** option causes used fonts to be listed on diagnostic output.

MOUSE

Clicking the right button will display the next page. Clicking the left button will display the previous page. Clicking the right button with the Shift key held down will display the next window full to the right. Clicking the left button with the Shift key held down will display the previous window full to the left. Clicking the middle button will display the next window full down. Clicking the middle button with the Shift key held down will display the next window full up.

KEYBOARD

You can exit the program by typing 'q', control-C, or control-D. You can move to the next page with 'n', 'f', or SPACE. You can move the the previous page with 'p', 'b', or control-H. You can move up a window-full with 'u', down with 'd', left with 'l', and right with 'r'. You can change the shrink factor by typing in the number (one or more digits), followed by 's'. If you type 's' without a number, the smallest factor that makes the entire page fit in the window will be used. You can force redisplay with control-L. You can move a relative number of pages by typing an optional '-', a number (one or more digits) and then carriage return or line feed. You can move to a specific page by typing a number (one or more digits) and then 'g'. You can move to the last page by typing 'g' without a number.

X DEFAULTS

Accepts the following defaults:

BorderWidth
Set the border width of the window.

ReverseVideo
If "on", reverse the definition of foreground and background color.

Foreground
Set the text/graphics color.

Background
Set the background color.

Border Set the border color.

Highlight
Set the page border color.

Mouse Set the mouse cursor color.

ENVIRONMENT

Uses the environment variable "DISPLAY" to specify which bit map display terminal to use.

SEE ALSO

X(1).

AUTHOR

Eric Cooper, CMU, did a version for direct output to a QVSS. Modified for X by Bob Scheifler, MIT Laboratory for Computer Science.

NAME

ximpv – Imprint (Impress) Previewer for the X Window System

SYNOPSIS

ximpv [=*geometry*] [-p#] [-rv] [-fg *color*] [-bg *color*] [-bd *color*] [-ms *color*] [*host:display*] file

DESCRIPTION

Ximpv is a program which runs under the X window system. It is used to preview images which is destined for an Imagen laser printer.

The *-p#* option, if used, will set the number of pages you can back up to #. Default is five pages. Zero (or no number) runs faster as the pages do not have to be transferred to disk.

If the *file* given to ximpv is correct a square will appear on the screen indicating text is about to appear. If no *file* is given stdin must be from a pipe or an error message is printed and the program aborted.

The pages of the file are displayed in the order. Only about 2/3 of a page can be displayed at once (this is because of aspect ratio differences).

ARGUMENTS

-bd *color*
Specify the border color.

-ms *color*
Specify the mouse color.

-fg *color*
Specify the foreground color.

-bg *color*
Specify the background color.

-bw *width*
Specify the width of the border.

-rv Cause *ximpv* to produce all output in black-on-white instead of white-on-black.

=geometry
The previewer window is created with the specified size specified by the geometry specification. See *X(1)* for details of this specification.

MOUSE

Clicking the right button will display the next window full, moving to the next page as needed. Clicking the middle button will move to the opposite end of the current page. Clicking the left button will display the previous window full, moving to the previous page as needed.

Clicking the right button with the Shift key held down will display the next window full to the right. Clicking the middle button with the Shift key held down will move to the opposite side of the current page. Clicking the left button with the Shift key held down will display the previous window full to the left.

KEYBOARD

The user may move up and down the page with the numeric pad keys:

	fine	medium	coarse
up page	7	8	9
down page	1	2	3

The numeric pad keys can also be used for horizontal motion:

	left	center	right
horizontal	4	5	6

You may also move forward or back in the document by using:
- for back a page,
up-arrow for back a window full,
. for forward a page,
down-arrow for forward a window full,
, or + for forward to next new page.
You may also move left and right in the document by small amounts using the left and right arrow keys.
The only other functional keys are the CNTRL **-D** key and the CNTRL **-C** key, which exit the program.

X DEFAULTS

Accepts the following defaults:

BorderWidth
Set the border width of the window.

ReverseVideo
If "on", reverse the definition of foreground and background color.

Foreground
Set the text/graphics color.

Background
Set the background color.

Border Set the border color.

Mouse Set the mouse cursor color.

SEE ALSO

X(1), xproof(1), xdvi(1)

ENVIRONMENT

Uses the environment variable "DISPLAY" to specify which bit map display terminal to use.

FILES

/usr/tmp/impvXXXXXX	circular buffer of screen images

SEE ALSO

X(1).

AUTHOR

Steven Sutphen and Ted Bentley, University of Alberta Changes and enhancements for X by Bob Scheifler, MIT Laboratory for Computer Science, and Jim Gettys, DEC, Project Athena.

NAME

xinit - X window system initializer

SYNOPSIS

xinit [[client] options] [-- [server] [display] options]

DESCRIPTION

Xinit is intended to be used when the X window system server is not run automatically from *init(8)*, and the window system must be started from a shell running on the display. This might be true, for example, if a normal login is run in a glass-tty emulator on a workstation console, so that different window systems can easily be run on the display at different times.

Xinit starts up the server and a single client application, which is typically *xterm(1)*. When the client eventually terminates, *xinit* automatically kills off the server and then itself terminates.

By default, *xinit* expects the server to exist in an executable named "X" in the search path, and for *xterm(1)* to also exist in the search path. It starts up the X server on display 0, and then starts up

```
        xterm =+1+1 -n login unix:0
```

A different client and/or server can be specified in the command line, and command line options can be passed to both the server and the client. The client and its options come first in the command line. The server and its options must be preceded by "--". If the first argument to xinit begins with '/' or a letter, it is taken to be the client program to use instead of xterm, and none of the default xterm options are used. Otherwise, the first and subsequent arguments are simply appended as further options to the default *xterm* command line.

Following the "--" argument, if the next argument begins with '/' or a letter, it is taken to be the server program to use instead of "X". If the next argument begins with a digit, it is taken to be the display number; otherwise display 0 is assumed. The remaining arguments are added as options to the server command line.

Examples:

```
xinit =80x65+10+10 -fn 8x13 -j -fg white -bg navy
xinit -e widgets -- Xsun -l -c
xinit rsh fasthost cpupig workstation:1 -- 1 -a 2 -t 5
```

AUTHOR

Copyright (c) 1986 by Massachusetts Institute of Technology.
See *X(1)* for a complete copyright notice.
Bob Scheifler, MIT Laboratory for Computer Science

SEE ALSO

X(8C), xterm(1)

NAME

xnwm - X window system manager process

SYNOPSIS

xnwm [-cmsnftv2] [*@border*] [*%iconDelta*]
[fm=*font*] [fi=*font*] [fs=*font*]
[l=*op*] [m=*op*] [r=*op*]
[*host*:*display*] [=*geometry*]

DESCRIPTION

The window manager is a process that allows the user of a display running the X window system to manipulate the windows on the screen. X implements the 'desktop model' of overlapping windows; *xnwm* allows windows to be moved, iconified, and resized, allows the order of the windows in the 'stack' of overlapping windows to be manipulated, and allows the keyboard focus to be attached to a window. X allows windows to contain other windows, but *xnwm* only manipulates the top-level windows and not any of the subwindows.

Xnwm takes arguments *host* and *display,* which refer the the host and display number. For example 'xnwm amadeus:1' would start up the window manager on display one on the machine amadeus. By default, *xnwm* uses the host and display number stored in the environment variable DISPLAY, and therefore they are not normally specified.

Xnwm has 2 modes of operation, 'normal' and 'popup', In normal mode *xnwm* creates a menu window across the top of the screen. To perform an action, you click any mouse button in the appropriate menu box and then click the same button in the window you wish to affect. *Xnwm* also reserves certain button/key combinations and interprets them as operations on existing windows. The key combination is specified in the command line with some subset of the options: **-c** (Control), **-m** (Meta), and **-s** (Shift). For example, if you specify the options **-cm** then the Control and Meta keys must be down at the time a mouse button is depressed. The option **-n** (None) means that no buttons need be held down. This is discouraged since it means that applications will never receive unshifted mouse clicks. If no combination is specified in the command line, Meta is assumed. Note: the key combination is not necessary when using functions from the menu; it is only needed with the assigned button functions to distinguish window manager operations from operations destined for the application running within the window.

The window manager normally takes control of the screen at various times to assure that the screen image remains correct while performing window manager operations. When this happens, requests from other applications are temporarily suspended until the window manager finishes the operation. The option **-f** (no freeze) disables this. If this option is specified, window outlines for *Move* and *Resize* will flicker rather than remaining solid, and the background behind popup windows (see later) will take longer to redraw.

The options **-t** (thin), **-v** (vertical), and **-2** (2 rows) control the format of the menu bar. In the absence of any of these, the menu extends across the entire screen. If the **-t** option is given, the menu bar will not extend fully across the screen; instead there will be room at the right (convenient for, for example, a clock window). If the **-v** option is used, the menu windows are stacked vertically instead of spread horizontally. The **-2** option causes the menu windows to be in two rows, allowing room for a terminal window the height of the screen while still allowing menu access. Either of the last two options automatically selects the **-t** option. The menu is located in the upper left corner of the screen by default, but its location can be set with the *=geometry* option as usual with X applications. (Notice that there is no size component, position information is used only.)

Xnwm will use reverse video for the menu, the cursor, icon text, and the frame around selected windows if the **-r** (reverse) option is used.

The border width around selected windows can be changed with the @ argument; the default is 5 pixels.

The default font for displaying text is "8x13". You can specify a different font with the **fm=** (Menu font), **fi=** (Icon font), and the **fs=** (Size window font) options.

Initially, the left, middle, and right mouse buttons are bound to the operations *Select, Raise,* and *Move.* You can change these bindings with the **l=** (left), **m=** (middle), and **r=** (right) arguments. Each should be followed by one of the letters "srmilzc", representing, respectively, *Select, Raise, Move, Iconify, Lower, resiZe,* and *Circulate.* They may also be followed by nothing, in which case no function is bound to that key.

Clicking any button that is not bound to the *Select* function in the background will cause the menu window to become visible if it has become covered by other windows. Double clicking the background will cause the menu to move back to its original position.

In popup mode, the menu window is not normally displayed, but instead 'pops up' when a particular button is pressed. To get popup mode, bind the letter "p" to any of the three buttons as described above. (You may also bind the other buttons as desired.) Whenever the bound button is clicked while the appropriate combination of control, meta, and shift keys is depressed, or any time a button that is not bound to the *Select* function is clicked in the background, the menu will appear beneath the cursor. You may then select any menu function you wish; after the operation is completed the menu will disappear. To make the menu disappear without performing any operation, just move the cursor out of the menu area. Note: the mouse button bound to the popup function may not be rebound using **Assign.** Using popup mode with complicated screen images and with no freeze (the **-f** option) may cause some difficulties if the menu obscures the image, since the applications will have to redraw their windows after the menu goes away.

The available commands are described below. For any of these commands, if you press a button to start a command, and then want to abort the command, simply press one of the other buttons before releasing the first button.

Select attaches the keyboard to a window, i.e., keyboard input will go to that window (hierarchy) even when the mouse is outside the window. It also **Raises** the selected window. Selecting the background will detach the keyboard from any window (actually, it attaches it to the background window). If no window is selected the keyboard input will go to the window which currently contains the mouse cursor. The selected window is highlighted by drawing a partial frame around the window. Selecting an icon allows the icon name to be edited: the delete key deletes the last character, control-U deletes the entire name, and other characters are appended to the current name. Typing a return restores the input focus to the most recent non-icon window selected.

Raise raises the window to the top of any stack of overlapping windows.

Move is used to move a window. If you apply it to a window, an outline will be moved with the mouse; when you release the button, the window will be moved.

(De)Iconify will make a window into an icon. If the mouse is moved more than a threshold amount, or this is the first time the window has been iconified, the icon will appear at the location on the screen where the button is released. Otherwise, the icon will reappear at its previous location. This threshold may be changed with the %*iconDelta* option. Giving a negative value will disable this effect. The default is 5 pixels. **(De)Iconify** will make the original window reappear at its former position on the screen if it is applied to an icon. The name displayed in the icon can be edited by **Selecting** the icon.

Lower will 'push' the window you point at to the bottom of any stack of overlapping windows.

Resize is used to resize a window by moving a corner or an edge. If you apply it to a window, a rubber banded outline of the window will be displayed and moving the mouse will change its size, leaving the opposite corner or other edges fixed. The corner or edge to be moved depends on the where the mouse is when the button is pressed. Imagine the window divided with grid of nine rectangles. If the mouse is in one of the four corner rectangles or the center rectangle, then the corner closest to the mouse will be moved; otherwise, the closest edge will be moved. When the button is released, the window will be resized.

Circulate causes the lowest window in the stack of overlapping windows to be **Raised** ; successive applications will reveal every window in turn.

Assign allows you to change the button bindings; to use it click any button in the Assign menu window and then click the same button in any other function to assign that function to that button. To remove the assignment from a button, double click the **Assign** window.

X DEFAULTS AND OPTION SUMMARY

MenuFont (fm=*name*)
Set the default font for the menu.

SizeFont (fs=*name*)
Set the default font for the size window.

IconFont (fi=*name*)
Set the default font for icons.

FrameWidth (*@value*)
Set the width of the frame around selected windows.

IconifyDelta (%*value*)
Set the threshold for moving icons.

ReverseVideo (-r)
Sets reverse video for the menu, icons, selection border, and cursor.

MenuFormat (-tv2)
Sets the format of the menu; should be some subset of **tv2** meaning thin, vertical, or 2 rows.

Freeze (-f)
If set to "off", disables *xnwm* taking control of the screen during operations.

KeyCombination (-csmln)
Sets the keys required to specify *xnwm* operations; should be some subset of **csmln** meaning control, shift, meta, lock, and none.

LeftButton (l=value)
Sets the default left button function; should be one of **srmilzcp**

MiddleButton (m=value)
Sets the default middle button function; should be one of **srmilzcp**

RightButton (r=value)
Sets the default right button function; should be one of **srmilzcp**

Geometry (={+-}xoff{+-}yoff)
Sets the location of the menu.

FILES

/usr/new/lib/X/font	directory of fonts

ENVIRONMENT

DISPLAY	- to get default host and display number

SEE ALSO

X(8C)

AUTHOR

Paul Asente, Stanford University, using some algorithms originally by Bob Scheifler, MIT Laboratory for Computer Science

NAME

xshell - X Window System, key/button command exec

SYNOPSIS

xshell [*options*] [*host:display*] ...

DESCRIPTION

Xshell is a program for starting up X applications with a single key or button stroke. It displays a scallop shell icon in which button and key presses stand for different commands. The user can bind a command string to any key or button by inserting a line like the following in his or her

```
xshell.action.keyname:   command to be exec'ed
```

Keynames are simply letters, numbers, and symbols as they appear on the keyboard (e.g. a, $, 9), or one of the following special names (taken from the X keyboard definitions):

```
KEYPAD0        FUNC1      E1
KEYPAD.        FUNC2      E2
ENTER          FUNC3      E3
KEYPAD1        FUNC4      E4
KEYPAD2        FUNC5      E5
KEYPAD3        FUNC6      E6
KEYPAD4        FUNC7      LEFTARROW
KEYPAD5        FUNC8      RIGHTARROW
KEYPAD6        FUNC9      DOWNARROW
KEYPAD,        FUNC10     UPARROW
KEYPAD7        FUNC11     SHIFT
KEYPAD8        FUNC12     CONTROL
KEYPAD9        FUNC13     LOCK
KEYPAD-        FUNC14     SYMBOL
PF1            FUNC15
PF2            FUNC16
PF3            FUNC17
PF4            FUNC18
LEFTBUTTON     FUNC19
MIDDLEBUTTON   FUNC29
RIGHTBUTTON
```

Thus, the following '.Xdefaults' definitions specify that the Left Button will spawn a terminal window, the Middle Button an editor, the Right Button a calculator, $ a Bourne shell, and # a superuser shell:

```
xshell.action.LeftButton:     xterm =80x65-0+0 -fn 6x10
xshell.action.MiddleButton:   xted =80x65+0-0
xshell.action.RightButton:    xterm =20x20-0-0 -fn 6x10 -e dc
xshell.action.$:              xterm =80x65+0+0 -fn 6x10 -e sh
xshell.action.#:              xterm =80x65+0+0 -fn 6x10 -e su
```

Xshell breaks the command string up into words by removing all white space (i.e. tabs and spaces) and uses the vfork() and execvp() system calls to spawn off the command. A more complicated parsing algorithm could easily be added, but the current method is adequate (and fast and memory efficient).

One thing to keep in mind is that *xshell* is NOT a window manager. It was written to make popping up frequently used utilities as painless as possible (how many times have you found that you need just 1 more window....). It might make a nice addition to some of the more verbose window managers, but it runs quite nicely as a separate program.

ARGUMENTS

Xshell is designed to be somewhat compatible with *xclock* in the arguments that it takes. However, *xshell* will allow you to abbreviate its longer flags to any length you chose. Thus, the –reverse flag can be spelled out, given as –rev, or even just –r:

–fg *color* On color displays, determines the color of the foreground.

–bg *color* On color displays, determines the color of the background.

–bd *color* On color displays, determines the color of the border.

–bw *pixels* Specify the width in pixels of the border around the *xshell* window.

–v[olume] *n*
Volume for calls to *XFeep*, used when errors (such as unbound key) are found.

–f[lash] *n* Number of times to flash the shell window to acknowledge a button or key press.

–d[elay] *n* One-hundredths of a second to wait between flashs (default is 5).

–r[everse] Reverse video (swap foreground and background).

–q[uiet] Do not 'feep' on errors (see volume).

–s[mall] Use a smaller (48x48) version of the shell icon. The default icon is 96x96.

=geometry
By default *xshell* will create a window the size of whatever icon you select; the standard X window geometry argument will override this. See *X(1)* for details.

host:display
specifies the display on which to put the *xshell* window. This overrides the DISPLAY environment variable.

X DEFAULTS

To make invoking *xshell* easier, each of the flags listed above may be specified in the user's

Foreground
gives the foreground color.

Background
gives the background color.

Border gives the border color.

BorderWidth
gives the border width.

ReverseVideo
if "on", the shell icon should be white on black instead of black on white.

Volume gives the volume to use in calls to XFeep().

Flash gives the number of times to flash the shell window to acknowledge key or button presses.

Delay gives hundredths of a second to wait in between flashes.

Quiet prevents *xshell* from feeping at you when you mistype.

IconSize if "small", a halfsize (48x48) version of the scallopshell is used.

WindowGeometry
gives the shell window size using standard X =WxH+X+Y notation.

ENVIRONMENT

DISPLAY

To get the default host and display number.

SEE ALSO

xwm(1), xnwm(1), X(1), execl(3), vfork(2)

DIAGNOSTICS

If **-quiet** is not given on the command line or "xshell.Quiet: on" does not appear in the user's *.Xdefaults*, *xshell* will 'feep' if a key or button is pressed for which there is no definition in the *.Xdefaults* file.

AUTHOR

Copyright 1985, Cognition Inc.

Jim Fulton (Cognition Inc.)

BUGS

Xshell uses the XGetDefault call to fetch the command string for a given key. Thus, you cannot bind the colon (":") character to a command.

A more 'user-friendly' interface could include dialog boxes that the user could pop up to type in a command directly so that a full shell doesn't have to be started. Then again, it is nice and compact now and if you really need to do that more than once you should use a real shell.

This program along with *xwm(1)* and *xnwm* have been mostly superceded by *uwm(1).*

NAME

xterm - X window system terminal emulator

SYNOPSIS

xterm [option] ...

DESCRIPTION

Xterm is the *X* window system terminal emulator. It attempts to emulate a DEC VT102 terminal (not yet completely implemented) to provide a standard terminal type for programs not aware of the window system directly. Under 4.3BSD and Ultrix 1.2, *xterm* supports the terminal resizing facilities built into the system.

When started, *xterm* pops a small window onto the upper left corner, with the size in characters and rows of the window as you size it. Once the window is created, a pseudo terminal is allocated and a shell is started on the slave side of the pty pair.

Xterm understands the following options:

-j *Xterm* will 'jump scroll'; when *xterm* falls behind scrolling the screen, it will move multiple lines up at once. This option is disabled by Tektronix mode. The VT100 escape sequences for smooth scroll can be used to enable/disable this feature from a program, or the 'Mode Menu' can be used to set it interactively.

-fn *font* The specified *font* will be used instead of the default font (which is vtsingle). Any fixed width font may be used.

-fb *font* The specified *font* will be used instead of the default bold font (which is vtbold). This font must be the same height and width as the normal font.

=geometry
: *Xterm* will take a normal X geometry specification. This takes the form of "*=widthxheight+xoff+yoff*". See *X(1)* for details of this specification.

host:display
: Normally, *xterm* gets the host and display number to use from the environment variable "DISPLAY". One can, however specify them explicitly. The *host* specifies which machine to create the window on, and the *display* argument specifies the display number. For example, "orpheus:1" creates a shell window on display one on the machine orpheus.

-n *windowname*
: Allows you to set the name of the window for use by a window manager.

-bw *borderwidth*
: Allows you to specify the width of the window border in pixels.

-b *border*
: *Xterm* maintains an inner border (distance between characters and the window's border) of one pixel. The **-b** option allows you to set the size of this border to *border.*

-rv The screen will be displayed with white characters on a black background, rather than the default black on white.

-fg *color*
: On color displays, determines the color of the text.

-bg *color*
: On color displays, determines the color of the background.

-bd *color*
: On color displays, determines the color of the border.

–cr *color*
: On color displays, determines the color of the text cursor; default is the text color.

–ms *color*
: On color displays, determines the color of the mouse cursor; default is the text cursor color.

–i asks *xterm* to maintain a bitmap icon, rather than relying on a window manager for an icon (see *xwm(1)*).

–t selects *Tektronix 4010* emulation in addition to normal vt102 emulation. In this mode, the default font is 6x10 and the default window size is 39x85. If a key is hit during Tektronix graphics output, the display may become garbled (just like a real Tektronix). The default screen size using the default font is one-fourth the resolution of a Tektronix 4010; therefore, some graphics may have discontinuities or may be suppressed entirely. Furthermore, the font initially selected approximately represents the standard Tektronix font, at best. If the window is subsequently enlarged or reduced, the font appears to shrink or grow, respectively. Resizing the window also affects resolution, and if the aspect ratio (height/width) is altered, Tektronix graphics will be restricted to the largest box with a 4010's aspect ratio that will fit in the window. This box is located in the upper left area of the window. Text which is part of Tektronix graphics output may not be cut (see MOUSE USAGE).

–e command arguments
: The specified *command* will be executed in the window, rather than starting a shell. The command and and optional arguments must appear last on the xterm command line.

–s When this option is specified, xterm no longer scrolls synchronously with the display. *Xterm* no longer attempts to keep the screen completely up to date while scrolling, but can then run faster when network latencies are very high. This is typically useful when using *xterm* across a very large internet or many hops.

–L indicates that *xterm* is being called by *init(8)*, and should presume that its file descriptors are already open on a slave pseudo-tty, and that *getty* should be run rather than the user's shell. This option should only be used by *init*.

MOUSE USAGE

When using the mouse to create the window, a cursor and a rubber banding box will outline where the window will be created on the display. If the left button is pressed, a HEIGHTxWIDTH (default 24x80) size window will be created. If the right button is pressed, a window the height of the display and WIDTH (default 80) characters wide will be created. If the center button is pressed and held down, the upper left hand corner of the window will be set to that point on the display, and (while continuing to depress the center button) an outline of the window will be displayed and the pop up window in the upper left corner of the screen will display the size in characters of the window.

Once the window is created, *xterm* allows you to save text and restore it within the same or other windows. The button functions are enabled when holding down the "shift" key. The left hand button takes the text from the cursor (at button release) through the end of line (including the new line), saves it in the global cut buffer, and immediately 'retypes' the line, inserting it as keyboard input. This provides a history mechanism. The center button is used to save text into the cut buffer. Move the cursor to beginning of the text, and then hold the button down while moving the cursor to the end of the region and releasing the button. The saved text will not include the character pointed by the mouse. Furthermore, it is not possible to cut text which was part of Tektronix graphics output. The right hand button 'types' the text from the cut buffer, inserting it as keyboard input. By cutting and pasting pieces of text without trailing new lines, you can take text from several places in different windows and form a command to the shell, for example, or take output from a program and insert it into your favorite editor. Since the cut buffer is globally shared among different applications, you should regard it as a 'file' whose contents you know. The terminal emulator and other text programs should be treating it as if it were a text file, i.e. the text is delimited by new lines.

X DEFAULTS

Xterm allows you to preset defaults in a customization file in your home directory, called *.Xdefaults.* The format of the file is "programname.keyword:string". See *X(1)* for more details. *Xterm* obeys the convention for 'MakeWindow' defaults. Keywords recognized by *xterm* are listed below.

JumpScroll
If "on" jump scroll is enabled.

BodyFont
Set the default font.

InternalBorder
Set the space between the text and window border. This is called padding above.

BorderWidth
Set the border width of the window.

ReverseVideo
If 'on', reverse the definition of foreground and background color.

Foreground
Set the text color.

Background
Set the background color.

Border Set the border color.

Cursor Set the text cursor color.

Mouse Set the mouse cursor color.

BitmapIcon
If 'on', use a bitmap icon for this window.

BoldFont
Specify a default bold font.

MODE MENU

Xterm has a menu for changing the modes of the terminal. The appearance of the menu is controlled by the defaults defined in the *XMenu(3x)* manual page. If you hold the "control" key down and press the middle mouse button, a pop-up menu appears. When you let up on the mouse button, the operation will be invoked. You can set the following modes of the emulator: "Smooth Scroll" vs. "Jump Scroll", "Reverse Video" vs. "Normal Video", "no wrap" vs. "auto wrap", "auto linefeed" vs. "normal linefeed", "application cursors" vs. "normal cursors", "application pad" vs. "numeric pad", and you can either "soft reset" or "hard reset" the emulator.

The scroll entry lets you control the scrolling behavior of the emulator as defined above. The video entry lets you change from normal to reverse video and back. The wrap entry lets you change to wrap at end of line or truncate at end of line. The linefeed entry lets you determine whether the emulator should provide a linefeed when the line wraps. The cursors entry lets you determine which escape sequences are generated by the cursor keys. The pad entry lets you determine if the numeric keypad should generate escape sequences or if it should generate numbers. The soft reset entry will reset scroll regions. This can be convenient when some program has left the scroll regions set incorrectly (often a problem when using VMS or

TOPS-20). The full reset entry will clear the screen, reset tabs to every eight columns, and reset the terminal modes to wrap and smooth scroll.

ENVIRONMENT

Xterm sets the environment variables "TERM" and "TERMCAP" properly for the size window you have created. It also uses and sets the environment variable "DISPLAY" to specify which bit map display terminal to use.

SEE ALSO

resize(1), xwm(1), X(1), pty(4), XMenu(3x)

DIAGNOSTICS

The **-d** flag turns on reporting of not understood escape sequences.

BUGS

Does not perfectly emulate a VT102 (though it is pretty close). While the 4010 emulation is as complete as we wish to make it, the Tektronix 4014 emulation is incomplete. Many applications will run. The display list for the Tektronix emulator needs more work.

AUTHORS

Mark Vandevoorde (MIT-Athena), Bob McNamara (DEC-MAD), Jim Gettys (MIT-Athena), Bob Scheifler (MIT-LCS), Doug Mink (SAO), Jordan Hubbard (Berkeley).

VMS and TOPS-20 are trademarks of Digital Equipment Corporation.

NAME

Xtext – routines to provide simple text output windows

SYNOPSIS

```
#include <X/Xlib.h>
#include <X/Xtext.h>

TextWindow *TextCreate(width, height, x, y, parent,
        fgpixel, bgpixel, bordercolor, fastscroll);
int height, width, x, y, bwidth, fgpixel, bgpixel, fastscroll;
Window parent;
char *fontname;
Pixmap bordercolor;

TextDestroy(t);
TextWindow *t;

TextClear(t);
TextWindow *t;

TextRedisplay(t);
TextWindow *t;

int TextEvent(t, e);
TextWindow *t;
XEvent *e;

TextPutString(t, str);
TextWindow *t;
char *str;

TextPutChar(t, ch);
TextWindow *t;
char ch;

TextPrintf(t, format [ , arg ] ... )
TextWindow *t;
char *format;
```

DESCRIPTION

These functions provide a simple interface to text output windows.

TextCreate creates a window that is *width* characters wide and *height* characters high. It is located with its upper left hand corner located at the point *x, y* in the window *parent.* The foreground (i.e. the characters) is in the color *fgpixel* and the background is the color *bgpixel.* The border is *bwidth* pixels wide and filled with the Pixmap *bordercolor.* If *fastscroll* is nonzero, text containing multiple newlines is displayed with a single jump scroll rather than with a single scroll for each newline.

The structure *TextWindow* is defined in */usr/include/X/Xtext.h.* The only field that should be of interest to most applications is *w,* the X Window id of the created window. This is quite useful if the application wishes to map the created window.

TextDestroy destroys the window described by its argument. The window is also destroyed automatically if the process creating it is terminated.

TextClear clears the window described by its argument.

TextRedisplay redisplays the window described by its argument.

TextEvent handles the event passed to it. It returns 0 if it was an event the library knows how to deal with, and 1 if it was an event of an unknown type; the latter should only happen if the application has changed the event mask for the window. Any event that the application receives that has as its *window* the window id of the text window should be passed to *TextEvent* for handling. Scrolling text generates an event per line of events, so the application should check for them frequently.

TextPutString prints its string in its window. The character '\n' (newline) is treated specially, and any other character is taken from the font. If the string contains multiple newlines, a single scroll is done for each line unless the *fastscroll* argument was non-zero in the call to *TextCreate.*

TextPutChar is similar to *TextPutString* but only prints a single character. Again, newline is treated specially.

TextPrintf is similar to the standard function *printf* except that it prints its result in the specified window. The resulting string is passed to *TextPutString.* See also the **BUGS** section at the end of this page.

SEE ALSO

printf(3S), xterm(1), X(8C)

AUTHOR

Paul Asente, Stanford University

BUGS

TextPrintf will truncate the output if the resulting string is more than 2048 characters long.

Since X operates asynchronously, it is possible to get way ahead of the server. This means that it may be quite a while between when a scroll happens on the screen and when *Xtext* gets around to filling in areas that couldn't be scrolled normally. This should only happen if the application issues a great many output requests very quickly, or if it doesn't get around to receiving the events *Xtext* needs to fill these areas in. Also, some strange TCP bugs are invoked if an application which has gotten far ahead of the X server is stopped (as with a control-Z).

NAME

xwd - X Window System, window image dumper.

SYNOPSIS

xwd [-debug] [-help] [-nobdrs] [-out *file*] [-z] [*host:display*]

DESCRIPTION

Xwd is an X Window System window image dumping utility. *Xwd* allows X users to store window images in a specially formated window dump file. This file can then be read by various other X utilities for redisplay, printing, editing, formatting, archiving, image processing etc.. The target window is selected by clicking the mouse in the desired window. The keyboard bell is rung once at the beginning of the dump and twice when the dump is completed. This is a preliminary version of what promises to be a very useful utility. The window dump file format is currently under development no guarantee of upward compatibility is made.

ARGUMENT SUMMARY

-help Print out the 'Usage:' command syntax summary.

-nobdrs This argument specifies that the window dump should not include the pixels that compose the X window border. This is useful in situations where you may wish to include the window contents in a document as an illustration.

-out *file* This argument allows the user to explicitly specify the output file on the command line. The default is to output to standard out.

-z This argument specifies that the output file should be in 'Z' pixmap format. The default is 'XY' pixmap format. 'Z' format is only valid on color displays.

host:display
This argument allow you to specify the host and display number on which to find the target window. For example 'xwd orpheus:1' would specify that the target window is on display '1' on the machine 'orpheus'. By default, *xwd* uses the host and display number stored in the environment variable DISPLAY, and therefore this argument is not normally specified.

ENVIRONMENT

DISPLAY
To get default host and display number.

FILES

XWDFile.h
X Window Dump File format definition file.

FUTURE PLANS

If time ever presents itself...

1. Install complete color support.
2. Completely rework the 'XWDFile' dump file format.
3. Completely rework the corresponding xwud window undumper program.

SEE ALSO

xwud(1), xpr(1), xdpr(1), X(1)

AUTHOR

Copyright 1985, Massachusetts Institute of Technology.

Tony Della Fera, Digital Equipment Corp., MIT Project Athena

NAME

xwininfo - X Window System, window information summarizer.

SYNOPSIS

xwininfo [-children] [-help] [-id *id*] [-int] [-root] [*host:display*]

DESCRIPTION

Xwininfo is a utility for displaying X window information summaries. All pertinent window information is displayed in an easily readable format. The user has the option of selecting the target window with the mouse (by clicking any mouse button in the desired window) or by specifying its' window id on the command line with the **-id** argument. There is also a special **-root** argument to quickly obtain information on X's root window. The following is a sample summary taken with the **-children** argument specified.

```
xwininfo ==> Please select the window you wish
         ==> information on by clicking the
         ==> mouse in that window.

xwininfo ==> Window name: ' X Root Window '
         ==> Window id: 0x10031
         ==> Parent window id: 0x0
         ==> Number of children: 13
         ====> Child window id: 0xb00046
         ====> Child window id: 0xb2004f
         ====> Child window id: 0x630051
         ====> Child window id: 0x5f0055
         ====> Child window id: 0x5c0058
         ====> Child window id: 0x55005c
         ====> Child window id: 0x53005e
         ====> Child window id: 0x510060
         ====> Child window id: 0x42000c
         ====> Child window id: 0x43000b
         ====> Child window id: 0x3d0011
         ====> Child window id: 0xa0028
         ====> Child window id: 0x500061
         ==> Associated window id: 0x0
         ==> Window type: IsOpaque
         ==> Window state: IsMapped
         ==> Upper left X: 0
         ==> Upper left Y: 0
         ==> Width: 1088
         ==> Height: 864
         ==> Border width: 0
         ==> Resize base width: 0
         ==> Resize base height: 0
         ==> Resize width increment: 1
         ==> Resize height increment: 1
         ==> Root absolute mouse X Position: 691
         ==> Root absolute mouse Y Position: 261
         ==> Target relative mouse X Position: 691
         ==> Target relative mouse Y Position: 261
```

ARGUMENT SUMMARY

-children
This argument specifies that *xwininfo* should list the window ids' of target window's children. Only the first level of the window hierarchy is shown (i.e., immediate children of the target window).

-help Print out the 'Usage:' command syntax summary.

-id *id* This argument allows the user to specify a target window *id* on the command line rather than using the mouse to select the target window. This is very useful in debugging X applications where the target window is not mapped to the screen or where the use of the mouse might be impossible or interfere with the application.

-int This argument specifies that all X window ids should be displayed as integer values. The default is to display them as hexadecimal values.

-root This argument specifies that X's root window is the target window. This is useful in situations where the root window is completely obscured.

host:display
This argument allow you to specify the host and display number on which to find the target window. For example 'xwininfo orpheus:1' would specify that the target window is on display '1' on the machine 'orpheus'. By default, *xwininfo* uses the host and display number stored in the environment variable DISPLAY, and therefore this argument is not normally specified.

ENVIRONMENT

DISPLAY
To get default host and display number.

SEE ALSO

X(1)

FUTURE PLANS

If time ever presents itself...

1. Provide a '-geometry' argument that prints out the window's dimensions in X window geometry format (i.e., =WxH+X+Y)

2. Provide a '-depth' argument that allows recursive traversal of the window hierarchy to some arbitrary depth.

AUTHOR

Copyright 1985, Massachusetts Institute of Technology.

Tony Della Fera, Digital Equipment Corp., MIT Project Athena

NAME

xwm - X Window System, window manager process

SYNOPSIS

xwm [-cfgmrsz] [*+function*] [*@delta*] [fn=*font*] [fi=*font*] [*host:display*]

DESCRIPTION

The window manager allows you to use the mouse to push a window to the top or bottom of the stack, turn a window into an icon, resize a window, move a window elsewhere on the screen, attach the keyboard to a window (hierarchy) and circulate the window hierarchy. The window manager only manipulates top-level windows (i.e., direct decendents of the root window), not their subwindows, so in the following, references to window refer only to top-level windows.

Since *xwm* does not have a window of its own it steals certain button/key combinations and interprets them as operations on existing windows. The key combination is specified on the command line with some subset of the options: **'c'** (control), **'s'** (shift), **'m'** (meta) and **'n'** (no-key). For example, if you specify the options **-cm** then the Control and Meta keys must be down at the time a mouse button is depressed. If no combination is specified in the command line, Meta is the default. If **'n'** is specified anywhere in the option list all keys will be ignored.

For each mouse button, a different command is performed depending on whether the button is 'clicked' or 'moved', i.e., whether the mouse is moved between the press and release of the button. Some actual movement is allowed before the mouse is really considered to have moved, the amount of movement is settable (see below). The mouse buttons perform the commands described below. For any of these commands, if you press a button to start a command, and then want to abort the command, simply press one of the other buttons before releasing the first button. As each command is being performed the mouse cursor will be changed to indicate which command is in effect.

If the left button is clicked in a window it will 'push' the window you are pointing at to the bottom of any stack of overlapping windows. If clicked on the root window a 'circulate down' operation will be performed on the root window moving the top most window in the hierarchy to the bottom. For any of these operations the mouse cursor will be a 'dot'.

The left button will also 'iconify' the window you point at if it is pressed down and then moved. When you release the button, the window will be made into an icon at the current mouse location. If the window being iconified has its own icon, then that icon will be used. If not *xwm* will create and maintain its own text icon using the name of the window as the initial text. For any of these operations the mouse cursor will be an 'icon' cursor.

The name displayed in an *xwm* owned text icon can be edited at any time by placing the mouse cursor in the icon and typing. Note: Modifying text displayed in an icon window will modify the name of the window associated with that icon. The delete key deletes the last character, Control-U deletes the entire name, any other printing characters are appended to the current name. When the mouse cursor in an *xwm* text icon it will be a 'text' cursor ('I bar' cursor).

If you click the middle button on an icon, the window you iconified will reappear in its previous position on the screen and the icon will disappear. For this operation the mouse cursor will be an 'arrow cross' cursor.

The middle button is used to resize a window by moving a corner or an edge. If you press it on a window, a rubber banded outline of the window will be displayed (and a grid if you specify the 'g' option explained below) and moving the mouse will change its size, leaving the opposite corner or other edges fixed. The corner or edge to be moved depends on the where the mouse is when the button is pressed. Imagine the window divided with grid of nine rectangles (the same grid that the 'g' option displays). If the mouse is in one of the four corner rectangles or the center rectangle, then the corner closest to the mouse will be moved; otherwise, the closest edge will be moved. When the button is released, the window will be resized. For these operations the mouse cursor will be an 'arrow cross' cursor.

The middle button can also be used to focus keyboard input to a specific window i.e., keyboard input will go to the specified window (hierarchy) even when the mouse is outside the window. If the 'f' option is specified clicking the middle button twice on a window will attach the keyboard to that window. Clicking the middle button on the background will detach the keyboard from any window (actually, it attaches it to the background window). For this operation the mouse cursor will be an 'arrow cross' cursor.

The right button, if clicked in a window, will 'pull' the window you are pointing at to the top of any stack of overlapping windows. If clicked on the root window a 'circulate up' operation will be performed moving the bottom most window in the hierarchy to the top. For these operations the mouse cursor will be a 'circle' cursor.

The right button will also move the window you are pointing at if it is pressed down and then moved. An outline of the window (and a grid if you specified the 'g' option) will appear, and will move with the mouse cursor. When you release the right button, the window will be moved to the current location of the outline. For this operation the mouse cursor will be a 'circle' cursor.

OPTION SUMMARY:

c The **'c'** (control) option specifies that the Control key must be held down for *xwm* to listen to mouse button operations.

f The **'f'** (focus) option specifies that a double-click on the middle button will focus keyboard input events to the specified window.

g The (grid) option turns on a tick-tack-toe like grid that will be displayed inside the 'window box' during window movement and resize operations.

m The **'m'** (meta) option specifies that the Meta key must be held down for *xwm* to listen to mouse button operations.

n The **'n'** (no-key) option specifies no keys may be down when performing mouse button operation.

r The **'r'** (reverse) option indicates that icons should be displayed as white text on a black background, rather than black text on a white background.

s The **'s'** (shift) options indicates that the Shift key must be held down for *xwm* to listen to mouse button operations.

z The **'z'** (zap) option turns on a special 'zap' effect that is intended to draw your attention to icons as they are created and windows as they are moved.

ARGUMENT SUMMARY:

+function
This argument allows you to specify a cursor display function. It should be followed by an integer specifying the code of the display function. See the Xlib document for details of available functions. The default function is GXcopy.

@delta This argument allows you to specify a mouse *delta* value. This value determines how far the mouse must move with a button down before the iconify, move and change operations begin. The default is 5 pixels. Note that if you define a large delta, you can still make fine adjustments by first moving far away and then moving back.

fn=*font* This argument allow you to specify a text *font* to be used in pop up information display. The default font is 6x10.

fi=*font* This argument allow you to specify an icon text font. The default font is 6x10.

host:display
This argument allow you to specify the the host and display number on which *xwm* will operate. For example 'xwm orpheus:1' would start up the window manager on display one on the machine orpheus. By default, *xwm* uses the host and display number stored in the environment variable DISPLAY, and therefore this argument is not normally specified. The window manager can be running anywhere, and you can run more than one manager per display (provided that they do not attempt to use the same mouse button / key combinations, see below).

X DEFAULTS

BodyFont
Set the default font for information display.

IconFont
Set the default font for text icons.

InternalBorder
Set the space between the text and window border in text icons.

BorderWidth
Set the border width of text icons.

ReverseVideo
Display text icons in reverse video?

ENVIRONMENT

DISPLAY
To get default host and display number.

SEE ALSO

X(1), X(8C)

AUTHOR

Copyright 1985, Massachusetts Institute of Technology.

Tony Della Fera, DEC MIT Project Athena

Based upon previous 'xwm' by Bob Scheifler, MIT Laboratory for Computer Science

XNS/COURIER USER CODE

J.Q. Johnson, Cornell

NAME

xnsftp –file transfer program

SYNOPSIS

ftp [**-v**] [**-d**] [**-i**] [**-n**] [**-g**] [**host**]

DESCRIPTION

Xnsftp is a user interface to the XNS Courier Filing protocol. The program allows a user to transfer files to and from a remote network site running Filing (version 4) server software, typically a Xerox file server.

The server host with which *xnsftp* is to communicate may be specified on the command line. If this is done, *xnsftp* will immediately attempt to establish a connection to a Filing server server on that host; otherwise, *xnsftp* will enter its command interpreter and await instructions from the user. When *xnsftp* is awaiting commands from the user the prompt "xnsftp>" is provided the user. The following commands are recognized by *xnsftp*:

! Invoke a shell on the local machine.

append *local-file* [*remote-file*]
Not yet implemented! Append a local file to a file on the remote machine. If *remote-file* is left unspecified, the local file name is used in naming the remote file. File transfer uses the current settings for *type*, *format*, *mode*, and *structure*.

ascii Set the file transfer *type* to network ASCII. This is the default type, and is appropriate for transferring 7-bit ascii text files.

bell Arrange that a bell be sounded after each file transfer command is completed.

binary Set the file transfer *type* to support binary image transfer. This is the appropriate type for transferring 8-bit binary data, e.g. Interlisp DCOM files.

bye Terminate the FTP session with the remote server and exit *xnsftp*.

cd *remote-directory*
Change the working directory on the remote machine to *remote-directory*.

close Terminate the FTP session with the remote server, and return to the command interpreter.

delete *remote-file*
Delete the file *remote-file* on the remote machine. If the remote file is a directory a confirmation will be required.

debug [*debug-value*]
Toggle debugging mode. If an optional *debug-value* is specified it is used to set the debugging level.

dir [*remote-directory*] [*local-file*]
Print a listing of the directory contents in the directory, *remote-directory*, and, optionally, placing the output in *local-file*. If no directory is specified, the current working directory on the remote machine is used. If no local file is specified, output comes to the terminal.

form *format*
Set the file transfer *form* to *format*. The default format, and the only one currently supported, is "file".

get *remote-file* [*local-file*]
Retrieve the *remote-file* and store it on the local machine. If the local file name is not specified, it is given the same name it has on the remote machine. The current settings for *type*, *form*, *mode*, and *structure* are used while transferring the file.

hash Toggle hash-sign ("#") printing for each data block transferred. Data blocks vary depending on implementation, but are typically 534 bytes long.

glob Toggle file name globbing. With file name globbing enabled, each local file or pathname is processed for *csh*(1) metacharacters. These characters include "*?[]~{}". Remote files specified in mutliple item commands, e.g. *mput*, are globbed by the remote server. With globbing disabled all files and pathnames are treated literally.

help [*command*]
Print an informative message about the meaning of *command*. If no argument is given, *xnsftp* prints a list of the known commands.

lcd [*directory*]
Change the working directory on the local machine. If no *directory* is specified, the user's home directory is used.

ls [*remote-directory*] [*local-file*]
Print an abbreviated listing (containing remote path names) of the contents of a directory on the remote machine. If *remote-directory* is left unspecified, the current working directory is used. If no local file is specified, the output is sent to the terminal.

mdelete *remote-files*
Delete the specified files on the remote machine. If globbing is enabled, the specification of remote files will first be expanded using *ls*.

mdir *remote-files local-file*
Obtain a directory listing of multiple files on the remote machine and place the result in *local-file*.

mget *remote-files*
Retrieve the specified files from the remote machine and place them in the current local directory. If globbing is enabled, the specification of remote files will first be expanding using *ls*. The local file names will be identical with the name attribute of the remote file names i.e. with the last component of the remote pathname.

mkdir *directory-name*
Make a directory on the remote machine.

mls *remote-files local-file*
Obtain an abbreviated listing of multiple files on the remote machine and place the result in *local-file*.

mode [*mode-name*]
Set the file transfer *mode* to *mode-name*. The default mode, and the only one currently supported, is "stream" mode.

mput *local-files*
Transfer multiple local files from the current local directory to the current working directory on the remote machine.

open *host* [*port*]
Establish a connection to the specified *host* Filing server. Note that *host* must be the Clearinghouse name of a Filing server, e.g. "cornellfs1:computer science:cornell-univ"; if the domain and organization components of the name are not specified, they default to the local domain and organization. If the *auto-login* option is on (default), *xnsftp* will also attempt to automatically log the user in to the Filing server (see below).

prompt Toggle interactive prompting. Interactive prompting occurs during multiple file transfers to allow the user to selectively retrieve or store files. If prompting is turned off (default), any *mget* or *mput* will transfer all files.

put *local-file* [*remote-file*]
: Store a local file on the remote machine. If *remote-file* is left unspecified, the local file name is used in naming the remote file. File transfer uses the current settings for *type*, *format*, *mode*, and *structure*.

pwd Print the name of the current working directory on the remote machine.

quit A synonym for bye.

rename [*from*] [*to*]
: Not yet implemented! Rename the file *from* on the remote machine, to the file *to*.

rmdir *directory-name*
: Delete a directory on the remote machine.

send *local-file* [*remote-file*]
: A synonym for put.

status Show the current status of *xnsftp*.

struct [*struct-name*]
: Set the file transfer *structure* to *struct-name*. By default "stream" structure is used. This is also the only structure currently supported.

trace Toggle packet tracing.

type [*type-name*]
: Set the file transfer *type* to *type-name*. If no type is specified, the current type is printed. The default type is network ASCII. Only ASCII and BINARY types are currently supported.

user *user-name* [*password*]
: Identify yourself to the remote Filing server. If the password is not specified and the server requires it, *xnsftp* will prompt the user for it (after disabling local echo). Unless *xnsftp* is invoked with "auto-login" disabled, this process is done automatically on initial connection to the Filing server. The user name should be a standard XNS Clearinghouse name or alias, e.g. "j.q. johnson:computer science:cornell-univ"; if the domain and organization components of the name are not specified, they default to the local domain and organization.

verbose Toggle verbose mode. In verbose mode, all responses from the Filing server are displayed to the user. In addition, if verbose is on, when a file transfer completes, statistics regarding the efficiency of the transfer are reported. By default, verbose is on.

? [*command*]
: A synonym for help.

Command arguments which have embedded spaces may be quoted with quote (") marks.

FILE NAMING CONVENTIONS

Files specified as arguments to *xnsftp* commands are processed according to the following rules.

1) If the file name "-" is specified, the **stdin** (for reading) or **stdout** (for writing) is used.

2) If the first character of the file name is "|", the remainder of the argument is interpreted as a shell command. *Xnsftp* then forks a shell, using *popen*(3) with the argument supplied, and reads (writes) from the stdout (stdin). If the shell command includes spaces, the argument must be quoted; e.g. ""| ls -lt"". A particularly useful example of this mechanism is: "dir |more".

3) Failing the above checks, if "globbing" is enabled, local file names are expanded according to the rules used in the *csh*(1); c.f. the *glob* command.

4) Remote file names whose first character is "/" (slash) are interpreted as absolute pathnames. Other remote file names are interpreted as pathnames relative to the current connected directory.

FILE TRANSFER PARAMETERS

The FTP specification specifies many parameters which may affect a file transfer. The *type* may be one of "ascii", "binary" (image), "ebcdic", and "local byte size" (for PDP-10's and PDP-20's mostly). *Xnsftp* supports the ascii and binary types of file transfer. ASCII type is appropriate for transferring text files; Unix EOL characters (\n) are translated to and from Xerox EOL characters (\r), Xerox left arrow characters are translated to underscore, etc. BINARY (image) type is appropriate for all other files.

Xnsftp supports only the default values for the remaining file transfer parameters: *mode*, *form*, and *struct*.

OPTIONS

Options may be specified at the command line, or to the command interpreter.

The **-v** (verbose on) option forces *xnsftp* to show all responses from the remote server, as well as report on data transfer statistics.

The **-n** option restrains *xnsftp* from attempting "auto-login" upon initial connection.

The **-i** option turns off interactive prompting during mutliple file transfers.

The **-d** option enables debugging.

The **-g** option disables file name globbing.

BUGS

Append and Rename are not yet implemented.

Many interesting features of the Filing protocol, e.g. serialized files and remote searches using the Find RPC, are not supported. Also, only version 4 of Filing is supported.

Aborting a file transfer does not work right; if one attempts this the connection to the remote server will likely have to be reopened.

NAME

CH_StringToName, CH_LookupAddr, CH_GetFirstCH, CH_GetOtherCH, CH_Enumerate, CH_EnumerateAliases – Clearinghouse support routines.

SYNOPSIS

```
#include <sys/types.h>            /* used by ns.h */
#include <netns/ns.h>             /* for sockaddr_ns */
#include <xnscourier/Clearinghouse2.h>
#include <xnscourier/CH.h>

Clearinghouse2_ObjectName CH_StringToName(string, defaults)
        char *string;
        Clearinghouse2_ObjectName *defaults;

struct xn_addr * CH_LookupAddr(pattern, property)
        Clearinghouse2_ObjectName pattern;
        Clearinghouse2_Property property;

CourierConnection * CH_GetFirstCH( )

CourierConnection * CH_GetOtherCH(conn, hint)
        CourierConnection *conn;
        Clearinghouse2_ObjectName hint;

int CH_Enumerate(pattern, property, eachName)
        Clearinghouse2_ObjectName pattern;
        Clearinghouse2_Property property;
        int (*eachName)();

CH_EnumerateAliases(pattern, eachName)
        Clearinghouse2_ObjectNamePattern pattern;
        int (*eachName)();
int eachName(name)
        Clearinghouse2_ObjectName name;
```

Link with *-lcourier*.

DESCRIPTION

These functions provide a convenient interface to the XNS Clearinghouse built on Courier remote procedure calls. They all require the Maryland XNS kernel.

Given a string in standard format (e.g. "jqj:Computer Science:cornell-univ"), *CH_StringToName* translates a string in standard format, e.g. "jqj:computer science:cornell-univ" into a Clearinghouse ObjectName struct. The storage for the resulting 3 fields is dynamically allocated via malloc(). If the argument string is incomplete, e.g. "jqj" or "::cornell-univ", the unspecified values are filled in from *defaults*. *Defaults* may be NULL, in which case 0-length strings are used as defaults.

Given a Clearinghouse three part name (possibly containing wild cards in the local object part) designating an addressable resouce on the net, and the property number on which a NetworkAddressList is expected to occur, *CH_LookupAddr* returns a pointer to an xn_addr structure associated with that name. Note that the xn_addr structure is statically allocated! If *property* is given as 0, then the "addressList" property (actually 4) is used; this is the property typically used for storing Clearinghouse addresses of objects. Returns 0 if any error occurs, if the name given is not registered, or if the name does not have the specified property. If a name has several network addresses (e.g. a gateway machine), only the first or primary address is returned; to obtain all addresses use the remote procedure *Clearinghouse2_RetrieveAddresses*. Users who require greater control than is provided by *CH_LookupAddress* should call *Clearinghouse2_RetrieveItem* directly. Example:

```
struct xn_addr * pvaxaddr;
static struct Clearinghouse2_ObjectName pvaxname =
      {"cornell-univ", "computer science", "pvax"};
```

```
pvaxaddr = CH_LookupAddr(pvaxname, 0);
```

The routine *CH_GetFirstCH* returns an XNS Courier connection to a nearby clearinghouse, useful for Clearinghouse remote procedure calls. Since the Clearinghouse is distributed, that instance of the CH may not contain the data desired; in such cases, a remote CH procedure call will return the error "WrongServer" with a hint as to the correct server, and the user may retry the operation after connecting (using *CH_GetOtherCH*) to the clearinghouse specified by the hint. For example:

```
conn = CH_GetFirstCH();
DURING objectname = Clearinghouse2_LookupObject(name, agent);
HANDLER {
   if (Exception.Code == Clearinghouse2_WrongServer) {
      hint = CourierErrArgs(Clearinghouse2_WrongServerArgs, hint);
      ch2conn = CH_GetOtherCH(conn, hint);
      CourierClose(conn);
      objectname = Clearinghouse2_LookupObject(name, agent);
   } else exit(1);
} END_HANDLER
```

CH_Enumerate and *CH_EnumerateAliases* each accept a pointer to a user-supplied function *eachProc*. This function is called once for each name in the local Clearinghouse satisfying the *pattern* (which may contain wildcards in its local object part only) supplied; *eachProc* is called with a single argument, the name of the current object. *CH_Enumerate* enumerates over all distinguished objects (i.e. no aliases) matching the specified pattern and having the specified *property*. To enumerate everything in a domain which has a given property, use "*" in the object portion of the pattern. To enumerate all names in a domain which match a given pattern, use the property value 0. Other useful property values are specified in *<courier/CHEntries.h>*.

CH_EnumerateAliases is similar to *CH_Enumerate* , except that *eachProc* is called once for each alias in the clearinghouse matching the specified pattern.

NOTES

A CourierConnection is an anonymous data structure used by the runtimes. Users should not dereference pointers to CourierConnection themselves.

Some useful definitions equivalent to those in the include file *<courier/Clearinghouse2.h>* include:

```
typedef struct {
        char *organization;
        char *domain;
        char *object;
} Clearinghouse2_ObjectName;

typedef unsigned long Clearinghouse2_Property;
```

FILES

```
/usr/local/lib/libcourier.a                      -lcourier library.
/usr/new/lib/xnscourier/clearinghouse.addresses  local clearinghouse address.
```

SEE ALSO

clearinghouse(3X)
"XNS Courier Under Unix".
"Clearinghouse Protocol," XSIS 078404 (April 1984).

DIAGNOSTICS

BUGS

Probably lots of them. This is an δ -test version of Courier support routines and is guaranteed to have bugs. Please report them to jqj@cornell.

In particular, since Packet Exchange is not yet working in the kernel, *CH_GetFirstCH* looks up the local clearinghouse address in a file rather than doing an expanding ring broadcast. This will be fixed soon.

CH_LookupAddr returns a pointer to a static data structure. The other routines use *malloc()* to dynamically allocate their data (you may use the routine *clear_Clearinghouse2_ObjectName* to free the strings allocated by *CH_StringToName*).

AUTHOR

J.Q. Johnson

NAME

(except) raise, raise_sys() – C exception handling

SYNOPSIS

#include <xnscourier/except.h>

raise(code, msg)
int code;
char *msg;

raise_sys()

cc ... –lexcept

EXTENDED C SYNTAX

DURING statement1 **HANDLER** statement2 **END_HANDLER**

E_RETURN(expression)

E_RETURN_VOID

DESCRIPTION

The macros and functions in this package provide a limited amount of exception handling for programming in C. They provide the ability to associate an exception handler to be invoked if an exception is raised during the execution of a statement.

C syntax is extended by several macros that allow the programmer to associate an exception handler with a statement. The "syntax" for this is:

DURING statement1 HANDLER statement2 END_HANDLER

Either or both statement may be a compound statement. If an exception is raised using the *raise*() function during *statement1* (or during any functions called by *statement1*), the stack will be unwound and *statement2* will be invoked in the current context. However, if the exception handler is redeclared in a *dynamically* enclosed statement, the current exception handler will be inactive during the execution of the enclosed statement.

During the execution of *statement2*, two predefined values may be used: *Exception.Code*, an integer, is the value of *code* passed to the *raise*() call which invoked the handler, and *Exception.Message* is the value of *msg*. It is up to the user to define the values used for the exception codes; by convention, small positive integers are interpreted as Unix error codes.

As an example of the use of this package, the following "toy" code computes the quotient of variables f1 and f2, unless f2 is 0.0:

```
DURING {
   if (f2 == 0.0)
        raise(DIVIDE_BY_ZERO, "Division by zero attempted");
   quotient = f1/ f2;
} HANDLER
   switch (Exception.Code) {
   case DIVIDE_BY_ZERO:
        return(HUGE);
        break;
   default:
        printf("Unexpected error %s\n", Exception.Message);
   }
```

END_HANDLER

If a handler does not want to take responsibility for an exception, it can "pass the buck" to the dynamically enclosing exception handler by use of the *RERAISE* macro, which simply raises the exception that invoked the handler. Of course, it is possible that there is no higher-level handler. The programmer can control the action in this case by setting the external int *ExceptMode* to some (bit-wise OR'd) combination of the following constants:

EX_MODE_REPORT	Print a message on stderr if an exception is not caught. If this is not set, no message is printed.
EX_MODE_ABORT	Calls the *abort*(3) routine if an exception is not caught. If this is not set, *exit*(3) is called, with the exception code as an argument.

The default value for *ExceptMode* is zero.

RESTRICTIONS

THESE RESTRICTIONS ARE IMPORTANT; YOU WILL SUFFER IF YOU DISOBEY THEM.

During the execution of *statement1,* no transfers out of the statement are allowed, except as noted here. Execution of a compound *statement1* must "run off the end" of the block. This means that *statement1* may not include a **return** or **goto**, or a **break** or **continue** that would affect a loop enclosing the *DURING ... END_HANDLER* block. The *statement1* may include a call to *raise*() (but not *RERAISE*), *exit*(3), and any statement at all may be used in a function called.

If you wish to use a **return** within *statement1*, you must instead use *E_RETURN()* to return a value, or *E_RETURN_VOID* if the enclosing function is declared **void**. These two macros may be used *only* in the (lexically) outermost *statement1* of a function, and nowhere else.

There are no restrictions on what may be done inside the *statement2* part of a handler block, except that it is subject to the above constraints if it is lexically enclosed in the *statement1* part of another handler.

As an aid to Unix programmers, the *raise_sys*() function is provided. It is used exactly as *raise*() is, except that it uses the global *errno*(3) to produce the exception code and message.

SEE ALSO

errno(3), setjmp(3)

AUTHOR

Jeffrey Mogul (Stanford)

BUGS

Due to a limitation of the *setjmp*(3) implementation, **register** variables which are actually stored in registers (and this is not always easy to determine, and especially is not portable) are restored to the values they had upon entering *statement1* when the handler (*statement2*) is invoked. All other data keeps whatever values they were assigned during the (interrupted) execution of *statement1*. A good rule to follow is that you should not rely on the values of variables declared **register** (in the current block) after an exception has been caught.

NAME

CourierOpen, CourierClose, BDTread, BDTwrite, BDTabort, BDTclosewrite – public run-times for Unix Courier

SYNOPSIS

#include <xnscourier/courier.h>
#include <xnscourier/courierconnection.h>

CourierConnection *CourierOpen(destaddr)
struct xn_addr *destaddr;

CourierClose(conn)
CourierConnection *conn;

int BDTread(conn, buffer, nbytes)
CourierConnection *conn;
char *buffer;
int nbytes;

int BDTwrite(conn, buffer, nbytes)
CourierConnection *conn;
char *buffer;
int nbytes;

BDTclosewrite(conn)
CourierConnection *conn;

BDTabort(conn)
CourierConnection *conn;

cc ... -lcourier

DESCRIPTION

These functions are part of the runtime library for XNS Courier remote procedure calls. They all require the Maryland XNS kernel.

CourierOpen attempts to open an SPP connection to the address specified. It returns 0 on failure.

CourierClose closes the SPP connection obtained by CourierOpen by means of the usual XNS 3-way END/END-REPLY handshake.

BDTread and *BDTwrite* are similar to *read(2)* and *write(2)* except that they accept a Courier connection instead of a file descriptor, and transmit or receive at most one SPP packet (maximum size is thus 534 bytes, which is also the recommended value of *nbytes*). These routines should be used only in a Courier server to perform a BDT data transfer, or in a Courier client from within a BDT callback routine.

BDTclosewrite and *BDTabort* provide a way for a BDT source (i.e. write) procedure to end a data transfer, either successfully or unsuccessfully respectively. In addition, *BDTabort* may be used to terminate a BDT sink (i.e. read) transfer.

FILES

SEE ALSO

all the Courier documentation

DIAGNOSTICS

None.

BUGS

Probably lots of them. Expanding ring broadcast is not yet implemented.