NAME

intro – introduction to miscellany

DESCRIPTION

This section describes miscellaneous facilities such as macro packages, character set tables, etc.

NAME

ascii – map of ASCII character set

SYNOPSIS

**cat /usr/pub/ascii**

DESCRIPTION

*Ascii* is a map of the ASCII character set, giving both octal and hexadecimal equivalents of each character, to be printed as needed. It contains:

```
|000 nul|001 soh|002 stx|003 etx|004 eot|005 enq|006 ack|007 bel|
|010 bs |011 ht |012 nl |013 vt |014 np |015 cr |016 so |017 si |
|020 dle|021 dc1|022 dc2|023 dc3|024 dc4|025 nak|026 syn|027 etb|
|030 can|031 em |032 sub|033 esc|034 fs |035 gs |036 rs |037 us |
|040 sp |041  ! |042  " |043  # |044  $ |045  % |046  & |047  ' |
|050  ( |051  ) |052  * |053  + |054  , |055  - |056  . |057  / |
|060  0 |061  1 |062  2 |063  3 |064  4 |065  5 |066  6 |067  7 |
|070  8 |071  9 |072  : |073  ; |074  < |075  = |076  > |077  ? |
|100  @ |101  A |102  B |103  C |104  D |105  E |106  F |107  G |
|110  H |111  I |112  J |113  K |114  L |115  M |116  N |117  O |
|120  P |121  Q |122  R |123  S |124  T |125  U |126  V |127  W |
|130  X |131  Y |132  Z |133  [ |134  \ |135  ] |136  ^ |137  _ |
|140  ` |141  a |142  b |143  c |144  d |145  e |146  f |147  g |
|150  h |151  i |152  j |153  k |154  l |155  m |156  n |157  o |
|160  p |161  q |162  r |163  s |164  t |165  u |166  v |167  w |
|170  x |171  y |172  z |173  { |174  | |175  } |176  ~ |177 del|

| 00 nul| 01 soh| 02 stx| 03 etx| 04 eot| 05 enq| 06 ack| 07 bel|
| 08 bs | 09 ht | 0a nl | 0b vt | 0c np | 0d cr | 0e so | 0f si |
| 10 dle| 11 dc1| 12 dc2| 13 dc3| 14 dc4| 15 nak| 16 syn| 17 etb|
| 18 can| 19 em | 1a sub| 1b esc| 1c fs | 1d gs | 1e rs | 1f us |
| 20 sp | 21  ! | 22  " | 23  # | 24  $ | 25  % | 26  & | 27  ' |
| 28  ( | 29  ) | 2a  * | 2b  + | 2c  , | 2d  - | 2e  . | 2f  / |
| 30  0 | 31  1 | 32  2 | 33  3 | 34  4 | 35  5 | 36  6 | 37  7 |
| 38  8 | 39  9 | 3a  : | 3b  ; | 3c  < | 3d  = | 3e  > | 3f  ? |
| 40  @ | 41  A | 42  B | 43  C | 44  D | 45  E | 46  F | 47  G |
| 48  H | 49  I | 4a  J | 4b  K | 4c  L | 4d  M | 4e  N | 4f  O |
| 50  P | 51  Q | 52  R | 53  S | 54  T | 55  U | 56  V | 57  W |
| 58  X | 59  Y | 5a  Z | 5b  [ | 5c  \ | 5d  ] | 5e  ^ | 5f  _ |
| 60  ` | 61  a | 62  b | 63  c | 64  d | 65  e | 66  f | 67  g |
| 68  h | 69  i | 6a  j | 6b  k | 6c  l | 6d  m | 6e  n | 6f  o |
| 70  p | 71  q | 72  r | 73  s | 74  t | 75  u | 76  v | 77  w |
| 78  x | 79  y | 7a  z | 7b  { | 7c  | | 7d  } | 7e  ~ | 7f del|
```

FILES

/usr/pub/ascii

## NAME

environ – user environment

## DESCRIPTION

An array of strings called the "environment" is made available by *exec*(2) when a process begins. By convention, these strings have the form "name=value". The following names are used by various commands:

| | |
|---|---|
| PATH | The sequence of directory prefixes that *sh*(1), *time*(1), *nice*(1), *nohup*(1), etc., apply in searching for a file known by an incomplete path name. The prefixes are separated by colons ( : ). *Login*(1) sets **PATH=:/bin:/usr/bin**. |
| HOME | Name of the user's login directory, set by *login*(1) from the password file *passwd*(4). |
| TERM | The kind of terminal for which output is to be prepared. This information is used by commands, such as *mm*(1), which may exploit special capabilities of that terminal. |
| TZ | Time zone information. The format is **xxx*n*zzz** where **xxx** is standard local time zone abbreviation, *n* is the difference in hours from GMT, and **zzz** is the abbreviation for the daylight-saving local time zone, if any; for example, **EST5EDT**. |

Further names may be placed in the environment by the *export* command and "name=value" arguments in *sh*(1), or by *exec*(2). It is unwise to conflict with certain shell variables that are frequently exported by **.profile** files: **MAIL**, **PS1**, **PS2**, **IFS**.

## SEE ALSO

env(1), login(1), sh(1), exec(2), getenv(3C), profile(4), term(5).

## NAME

eqnchar – special character definitions for eqn and neqn

## SYNOPSIS

**eqn /usr/pub/eqnchar** [ files ] **|** **troff** [ options ]

**neqn /usr/pub/eqnchar** [ files ] **|** **nroff** [ options ]

## DESCRIPTION

*Eqnchar* contains *troff*(1) and *nroff* character definitions for constructing characters that are not available on the Wang Laboratories, Inc. C/A/T phototypesetter. These definitions are primarily intended for use with *eqn*(1) and *neqn*; *eqnchar* contains definitions for the following characters:

| | | | | | |
|---|---|---|---|---|---|
| *ciplus* | ⊕ | \| \| | ‖ | *square* | |
| *citimes* | ⊗ | *langle* | ⟨ | *circle* | ○ |
| *wig* | ∼ | *rangle* | ⟩ | *blot* | |
| *-wig* | ≃ | *hbar* | ℏ | *bullet* | • |
| *>wig* | ≳ | *ppd* | ⊥ | *prop* | ≈ |
| *<wig* | ≲ | *<->* | ↔ | *empty* | ∅ |
| *=wig* | ≅ | *<=>* | ⇔ | *member* | ∈ |
| *star* | * | *\|<* | <⃒ | *nomem* | ∉ |
| *bigstar* | ✳ | *\|>* | >⃒ | *cup* | ∪ |
| *=dot* | ≐ | *ang* | ∠ | *cap* | ∩ |
| *orsign* | \/ | *rang* | ∟ | *incl* | ⊑ |
| *andsign* | /\ | *3dot* | ⋮ | *subset* | ⊂ |
| *=del* | ≜ | *thf* | ∴ | *supset* | ⊃ |
| *oppA* | ∀ | *quarter* | ¼ | *!subset* | ⊆ |
| *oppE* | ∃ | *3quarter* | | *!supset* | ⊇ |
| *angstrom* | Å | *degree* | ° | *scrL* | ℓ |
| *==<* | ≤ | *==>* | ≥ | | |

## FILES

/usr/pub/eqnchar

## SEE ALSO

eqn(1), nroff(1), troff(1).

## NAME

fcntl – file control options

## SYNOPSIS

**#include <fcntl.h>**

## DESCRIPTION

The *fcntl*(2) function provides for control over open files. This include file describes *requests* and *arguments* to *fcntl* and *open*(2).

```
/* Flag values accessible to open(2) and fcntl(2) */
/*  (The first three can only be set by open) */
#define O_RDONLY  0
#define O_WRONLY  1
#define O_RDWR    2
#define O_NDELAY  04          /* Non-blocking I/O */
#define O_APPEND  010         /* append (writes guaranteed at the end) */
#define O_DIRECT  0100000     /* Direct I/O */

/* Flag values accessible only to open(2) */
#define O_CREAT   00400       /* open with file create (uses third open arg)*/
#define O_TRUNC   01000       /* open with truncation */
#define O_EXCL    02000       /* exclusive open */

/* fcntl(2) requests */
#define F_DUPFD   0           /* Duplicate fildes */
#define F_GETFD   1           /* Get fildes flags */
#define F_SETFD   2           /* Set fildes flags */
#define F_GETFL   3           /* Get file flags */
#define F_SETFL   4           /* Set file flags */
```

## SEE ALSO

fcntl(2), open(2).

NAME

man – macros for formatting entries in this manual

SYNOPSIS

**nroff –man** files

DESCRIPTION

These *troff*(1) macros are used to lay out the format of the entries of this manual. A skeleton entry may be found in the file **/usr/man/u_man/man0/skeleton**. These macros are used by the *man*(1) command.

Any *text* argument below may be one to six "words". Double quotes ("") may be used to include blanks in a "word". If *text* is empty, the special treatment is applied to the next line that contains text to be printed. For example, **.I** may be used to italicize a whole line, or **.SM** followed by **.B** to make small bold text. By default, hyphenation is turned off for *nroff*, but remains on for *troff*.

Type font and size are reset to default values before each paragraph and after processing font- and size-setting macros, e.g., **.I**, **.RB**, **.SM**. Tab stops are neither used nor set by any macro except **.DT** and **.TH**.

Default units for indents *in* are ens. When *in* is omitted, the previous indent is used. This remembered indent is set to its default value (7.2 ens in *troff*, 5 ens in *nroff*–this corresponds to 0.5′ ′ in the default page size) by **.TH**, **.P**, and **.RS**, and restored by **.RE**.

| | |
|---|---|
| .TH *t s c n* | Set the title and entry heading; *t* is the title, *s* is the section number, *c* is extra commentary, e.g., "local", *n* is new manual name. Invokes .DT (see below). |
| .SH *text* | Place subhead *text*, e.g., SYNOPSIS, here. |
| .SS *text* | Place sub-subhead *text*, e.g., **Options**, here. |
| .B *text* | Make *text* bold. |
| .I *text* | Make *text* italic. |
| .SM *text* | Make *text* 1 point smaller than default point size. |
| .RI *a b* | Concatenate roman *a* with italic *b*, and alternate these two fonts for up to six arguments. Similar macros alternate between any two of roman, italic, and bold: .IR .RB .BR .IB .BI |
| .P | Begin a paragraph with normal font, point size, and indent. **.PP** is a synonym for **.P**. |
| .HP *in* | Begin paragraph with hanging indent. |
| .TP *in* | Begin indented paragraph with hanging tag. The next line that contains text to be printed is taken as the tag. If the tag does not fit, it is printed on a separate line. |
| .IP *t in* | Same as .TP *in* with tag *t*; often used to get an indented paragraph without a tag. |
| .RS *in* | Increase relative indent (initially zero). Indent all output an extra *in* units from the current left margin. |
| .RE *k* | Return to the *k*th relative indent level (initially, *k*=1; *k*=0 is equivalent to *k*=1); if *k* is omitted, return to the most recent lower indent level. |
| .PM *m* | Produces proprietary markings; where *m* may be **P** for **PRIVATE**, **N** for **NOTICE**, **BP** for **BELL LABORATORIES PROPRIETARY**, or **BR** for **BELL LABORATORIES RESTRICTED**. |
| .DT | Restore default tab settings (every 7.2 ens in *troff*, 5 ens in *nroff*). |
| .PD *v* | Set the interparagraph distance to *v* vertical spaces. If *v* is omitted, set the interparagraph distance to the default value (0.4v in *troff*, 1v in *nroff*). |

The following *strings* are defined:

| | |
|---|---|
| *R | in *troff*, **(Reg.)** in *nroff*. |
| *S | Change to default type size. |
| *(Tm | Trademark indicator. |

The following *number registers* are given default values by **.TH**:

| | |
|---|---|
| IN | Left margin indent relative to subheads (default is 7.2 ens in *troff*, 5 ens in *nroff*). |
| LL | Line length including IN. |
| PD | Current interparagraph distance. |

## CAVEATS

In addition to the macros, strings, and number registers mentioned above, there are defined a number of *internal* macros, strings, and number registers. Except for names predefined by *troff* and number registers **d**, **m**, and **y**, all such internal names are of the form *XA*, where *X* is one of ), ], and }, and *A* stands for any alphanumeric character.

If a manual entry needs to be preprocessed by *cw*(1), *eqn*(1) (or *neqn*), and/or *tbl*(1), it must begin with a special line (described in *man*(1)), causing the *man* command to invoke the appropriate preprocessor(s).

The programs that prepare the Table of Contents and the Permuted Index for this Manual assume the *NAME* section of each entry consists of a single line of input that has the following format:

name[, name, name . . .] \- explanatory text

The macro package increases the inter-word spaces (to eliminate ambiguity) in the *SYNOPSIS* section of each entry.

The macro package itself uses only the roman font (so that one can replace, for example, the bold font by the constant-width font–see *cw*(1)). Of course, if the input text of an entry contains requests for other fonts (e.g., **.I**, **.RB**, **\fI**), the corresponding fonts must be mounted.

## FILES

/usr/lib/tmac/tmac.an
/usr/lib/macros/cmp.[nt].[dt].an
/usr/lib/macros/ucmp.[nt].an
/usr/man/[ua]_man/man0/skeleton

## SEE ALSO

man(1), nroff(1).

## BUGS

If the argument to .TH contains *any* blanks and is *not* enclosed by double quotes (""), there will be bird-dropping-like things on the output.

## NAME

math – math functions and constants

## SYNOPSIS

**#include <math.h>**

## DESCRIPTION

This file contains declarations of all the functions in the Math Library (described in Section 3M), as well as various functions in the C Library (Section 3C) that return floating-point values.

It defines the structure and constants used by the *matherr*(3M) error-handling mechanisms, including the following constant used as an error-return value:

| | |
|---|---|
| HUGE | The maximum value of a single-precision floating-point number. |

The following mathematical constants are defined for user convenience:

| | |
|---|---|
| M_E | The base of natural logarithms (*e*). |
| M_LOG2E | The base-2 logarithm of *e*. |
| M_LOG10E | The base-10 logarithm of *e*. |
| M_LN2 | The natural logarithm of 2. |
| M_LN10 | The natural logarithm of 10. |
| M_PI | The ratio of the circumference of a circle to its diameter. (There are also several fractions of its reciprocal and its square root.) |
| M_SQRT2 | The positive square root of 2. |
| M_SQRT1_2 | The positive square root of 1/2. |

For the definitions of various machine-dependent "constants," see the description of the <*values.h*> header file.

## FILES

/usr/include/math.h

## SEE ALSO

intro(3), matherr(3M), values(5).

## NAME

mm – the MM macro package for formatting documents

## SYNOPSIS

**mm** [ options ] [ files ]

**nroff –mm** [ options ] [ files ]

**nroff –cm** [ options ] [ files ]

## DESCRIPTION

This package provides a formatting capability for a very wide variety of documents. It is the standard package used by the BTL typing pools and documentation centers. The manner in which a document is typed in and edited is essentially independent of whether the document is to be eventually formatted at a terminal or is to be phototypeset. See the references below for further details.

The **–mm** option causes *nroff* and *troff*(1) to use the non-compacted version of the macro package, while the **–cm** option results in the use of the compacted version, thus speeding up the process of loading the macro package.

## FILES

| | |
|---|---|
| /usr/lib/tmac/tmac.m | pointer to the non-compacted version of the package |
| /usr/lib/macros/mm[nt] | non-compacted version of the package |
| /usr/lib/macros/cmp.[nt].[dt].m | compacted version of the package |
| /usr/lib/macros/ucmp.[nt].m | initializers for the compacted version of the package |

## SEE ALSO

mm(1), mmt(1), nroff(1).
*MM-Memorandum Macros* by D. W. Smith and J. R. Mashey.
*Typing Documents with MM* by D. W. Smith and E. M. Piskorik.

## NAME

modemcap – smart modem capability data base

## SYNOPSIS

/usr/lib/uucp/modemcap

## DESCRIPTION

*Modemcap* describes the call placing protocol of smart modems. Operating system *uucp (1C)* and *dial (3C)* accept a reference to a *modemcap* entry in place of an automatic call unit reference in **/usr/lib/uucp/L–devices**. Each entry describes a single modem in a specific configuration.

*Modemcap* is a text file. Lines that begin with a pound sign (#) are ignored. Other lines make up descriptions.

Each description begins on a new line. The beginning of the description is a list of its names, separated by vertical bars ( | ). Any of the names, which must not begin with **cua**, can be used in place of the call unit name in **/usr/lib/uucp/L–devices**.

The rest of the description is a list of capabilities, separated by colons (:). If a description extends over more than one line, each line except the last must end with a backslash (\\). (The continuation is normally entered as colon-backslash-newline-tab-colon: this produces a single invalid capability, which is ignored.) Here is an example:

```
# bizcomp 1012 - option switch 9 down
bz | bizcomp bizcomp 1012:
        :a1=NO ANSWER:b1=NO DIAL TONE:b2=NO ANSWER:c1=1:c2=2:\
        :c7=7:d1#1:d5#5:eh=\r:ph=\02D:ps=\02:pw= 72:\
        :sa=A:sq=Q:sv=V:sx=X:sz=Z:wp=\r:\
        :pl=szd5wpd1svwpsqwpsxwpd1phwpc7b1wpc2a1c1b2d1:
```

Each capability has three parts:

1. The two-character name of the capability.
2. An pound sign (#) or equals sign (=). A pound sign indicates a numeric capability. An equals sign indicates a string capability.
3. The capability value. For a numeric capability, the value is the number that immediately follows the pound sign. For a string capability, the value is the string of characters, including blanks, between the equals sign and the colon that ends the capability. (If a colon is part of the value, it must be expressed as an octal sequence; see below.) In a string capability, the following sequences stand for single characters:

| | |
|---|---|
| \*xxx* | (where *xxx* is one to three octal digits) the character whose octal value is *xxx* |
| \\072 | colon (:) |
| \\200 | null (\\000 doesn't work) |
| \\E | escape (\\033) |
| \\n | newline (\\012) |
| \\r | return (\\015) |
| \\t | tab (\\011) |
| \\b | backspace (\\010) |
| \\f | formfeed (\\014) |
| ^*x* | control-*x* |

There are four kinds of capabilities: the place call capability, basic features capabilities, the send phone number capability, and send/receive capabilities. Only the place call capability is mandatory.

**Place Call Capability**

**pl** String capability. Controls the use of the other capabilities. The value of the string is a procedure made up of the other capabilities. A communication program works through **pl's** value, using each capability as it is encountered; a limited control of execution flow is provided by some special capabilities.

**Basic Features Capabilities**

Basic features capabilities specify strings used to command basic features of the modem. These capabilities never appear in the **pl** value, but are implied by other capabilities. The capability descriptions indicate which capabilities use basic features capabilities and what happens when basic features capabilities are undefined.

**ps** Primary command start; string capability. The **ps** capability specifies the characters that precede modem commands, if required. Used by **s***x* capability.

es ~~**pe**~~ Primary command end; string capability. The es ~~**pe**~~ capability specifies the characters that must follow modem commands, if required. Used by **s***x* capability.

**eh** End phone number; string capability. Used by **ph** capability.

**pa** Pause in phone number; string capability. Used by **ph** capability.

**pw** Pause in phone number and wait for dial tone; string capability. Used by **ph** capability.

**Send Phone Number Capability**

**ph** String capability. In a single *write*(2), send a string with three parts:

1. The **ph's** capability's own value.
2. The phone number as ASCII digits. Whenever the modem should pause, send the value of the **pa** capability, if defined. Whenever the modem should pause and wait for a dial tone, send the value of the **pw** capability, if defined.
3. The value of the **eh** capability, if defined.

**Send/Receive Capabilities**

Send/receive capabilities are different from other capabilities in their naming convention. The first character of the capability name tells the kind of capability. The second character of the name is chosen arbitrarily from the lowercase letters and digits and identifies the particular capability from others of the same kind.

**t***x* String capability. Send the value to the modem.

**s***x* String capability. In a single *write*, send a command to the modem. The command has three parts:

1. The value of the **ps** capability, if defined.
2. The **s***x's* cpability's own value.
3. The value of the **pe** capability, if defined.

**d***x* Numeric capability. Delay for the number of seconds specified in the value.

**w***x* String capability; value must be a single character. Wisk through input from modem until the value is read. Put input, up to but not including the terminating character, in the wisk buffer, replacing the previous contents.

**c***x* String capability. Compare value with contents of the wisk buffer. Set the comparison flag to EQUAL if they match, NOT_EQUAL otherwise. Do not modify the comparison flag until you execute another **c***x*.

**m***x* Numeric capability. Skip on EQUAL. If the comparison flag is EQUAL the next *n* instructions in the **pl** value are skipped, where *n* is the value of **m***x*.

**n***x* Numeric capability. Skip on NOT_EQUAL. If the comparison flag is NOT_EQUAL the next *n* instructions in the **pl** value are skipped, where *n* is the value of **n***x*.

**a***x* String capability. Abort on EQUAL. If the comparison flag is EQUAL abort the phone call. If debug output is specified, print the value of the **a***x* capability.

**b***x* String capability. Abort on NOT_EQUAL. If the comparison flag is NOT_EQUAL abort the phone call. If debug output is specified, print the value of the **b***x* capability.

EXAMPLE

The Bizcomp 1012 example above assumes that the modem's switch 9 (configuration: TERMINAL/COMPUTER) is down (COMPUTER). With this setting, the modem has the following characteristics:

- Commands to the modem must be preceded by an STX (\002) and followed by a CR (\r). This prevents normal data transmissions from being taken for modem commands.
- The modem's messages to the computer are terse. The following two-character sequences are diagnostics.

| | |
|---|---|
| 1 CR | connection made |
| 2 CR | no connection or no answer |
| 7 CR | dial tone detected |

A CR is a command prompt. A communication program that uses the Bizcom 1012 *modemcap* entry follows the following procedure:

1. (szd5wpdl) Send an STX-Z-CR, resetting the modem. Wait five seconds, then read the resulting CR. Wait another one second.
2. (svwpsqwpsxwpdl) Send an STX-V-CR (select tone dialing); read the resulting CR. Send an STX-Q-CR (toggle busy detection); read the resulting CR. Send an STX-X-CR (select transparent data mode); read the resulting CR. Wait one second.
3. (ph) Send an STX-D, then the phone number. The phone number should include a colon (:) whenever the modem should pause to listen for another dial tone. The description lacks a **pa** capability, so there is no way to pause without waiting for a dial tone.
4. (wpc7bl) Read until the next CR. If the input isn't "7", abort with the debug message "NO DIAL TONE".
5. (wpc2alclb2) Read until the next CR. If the input is "2", abort with the debug message "NO ANSWER". Otherwise, if the input isn't "1", abort with the debug message "NO ANSWER".
6. (dl) Wait one second. The connection is established.

SEE ALSO

dial(3C), uucp(1C).

## NAME

mptx – the macro package for formatting a permuted index

## SYNOPSIS

**nroff –mptx** [ options ] [ files ] [ options ] [ files ]

## DESCRIPTION

This package provides a definition for the **.xx** macro used for formatting a permuted index as produced by *ptx*(1). This package does not provide any other formatting capabilities such as headers and footers. If these or other capabilities are required, the *mptx* macro package may be used in conjuction with the *MM* macro package. In this case, the **–mptx** option must be invoked *after* the **–mm** call. For example:

```
nroff -cm -mptx file
```

or

```
mm -mptx file
```

## FILES

| | |
|---|---|
| /usr/lib/tmac/tmac.ptx | pointer to the non-compacted version of the package |
| /usr/lib/macros/ptx | non-compacted version of the package |

## SEE ALSO

mm(1), nroff(1), ptx(1), mm(5).

## NAME

mv – a troff macro package for typesetting view graphs and slides

## SYNOPSIS

**mvt** [ **–a** ] [ options ] [ files ]

**troff** [ **–a** ] [ **–rX1** ] **–mv** [ options ] [ files ]

## DESCRIPTION

This package makes it easy to typeset view graphs and projection slides in a variety of sizes. A few macros (briefly described below) accomplish most of the formatting tasks needed in making transparencies. All of the facilities of *troff*(1), *cw*(1), *eqn*(1), and *tbl*(1) are available for more difficult tasks.

The output can be previewed on most terminals, and, in particular, on the Tektronix 4014, as well as on the Versatec printer. For these two devices, specify the **–rX1** option (this option is automatically specified by the *mvt* command–q.v.–when that command is invoked with the **–T4014** or **–Tvp** options). To preview output on other terminals, specify the **–a** option.

The available macros are:

| | | |
|---|---|---|
| **.VS** | [*n*] [*i*] [*d*] | Foil-start macro; foil size is to be 7′ ′ ×7′ ′ ; *n* is the foil number, *i* is the foil identification, *d* is the date; the foil-start macro resets all parameters (indent, point size, etc.) to initial default values, except for the values of *i* and *d* arguments inherited from a previous foil-start macro; it also invokes the **.A** macro (see below).<br><br>The naming convention for this and the following eight macros is that the first character of the name (**V** or **S**) distinguishes between view graphs and slides, respectively, while the second character indicates whether the foil is square (**S**), small wide (**w**), small high (**h**), big wide (**W**), or big high (**H**). Slides are "skinnier" than the corresponding view graphs: the ratio of the longer dimension to the shorter one is larger for slides than for view graphs. As a result, slide foils can be used for view graphs, but not vice versa; on the other hand, view graphs can accommodate a bit more text. |
| **.Vw** | [*n*] [*i*] [*d*] | Same as **.VS**, except that foil size is 7′ ′ wide × 5′ ′ high. |
| **.Vh** | [*n*] [*i*] [*d*] | Same as **.VS**, except that foil size is 5′ ′ ×7′ ′ . |
| **.VW** | [*n*] [*i*] [*d*] | Same as **.VS**, except that foil size is 7′ ′ ×5.4′ ′ . |
| **.VH** | [*n*] [*i*] [*d*] | Same as **.VS**, except that foil size is 7′ ′ ×9′ ′ . |
| **.Sw** | [*n*] [*i*] [*d*] | Same as **.VS**, except that foil size is 7′ ′ ×5′ ′ . |
| **.Sh** | [*n*] [*i*] [*d*] | Same as **.VS**, except that foil size is 5′ ′ ×7′ ′ . |
| **.SW** | [*n*] [*i*] [*d*] | Same as **.VS**, except that foil size is 7′ ′ ×5.4′ ′ . |
| **.SH** | [*n*] [*i*] [*d*] | Same as **.VS**, except that foil size is 7′ ′ ×9′ ′ . |
| **.A** | [*x*] | Place text that follows at the first indentation level (left margin); the presence of *x* suppresses the line spacing from the preceding text. |
| **.B** | [*m* [*s*] ] | Place text that follows at the second indentation level; text is preceded by a mark; *m* is the mark (default is a large bullet); *s* is the increment or decrement to the point size of the mark with respect to the *prevailing* point size (default is 0); if *s* is 100, it causes the point size of the mark to be the same as that of the *default* mark. |
| **.C** | [*m* [*s*] ] | Same as **.B**, but for the third indentation level; default mark is a dash. |
| **.D** | [*m* [*s*] ] | Same as **.B**, but for the fourth indentation level; default mark is a small bullet. |
| **.T** | *string* | *String* is printed as an over-size, centered title. |
| **.I** | [*in*] [*a* [*x*] ] | Change the current text indent (does not affect titles); *in* is the indent (in inches unless dimensioned, default is 0); if *in* is signed, it is an increment or decrement; the presence of *a* invokes the **.A** macro (see below) and passes *x* (if any) to it. |

**.S** [*p*] [*l*] Set the point size and line length; *p* is the point size (default is "previous"); if *p* is 100, the point size reverts to the *initial* default for the current foil-start macro; if *p* is signed, it is an increment or decrement (default is 18 for **.VS**, **.VH**, and **.SH,** and 14 for the other foil-start macros); *l* is the line length (in inches unless dimensioned; default is 4.2′ ′ for **.Vh**, 3.8′ ′ for **.Sh**, 5′ ′ for **.SH**, and 6′ ′ for the other foil-start macros).

**.DF** *n f* [*n f*...]
Define font positions; may not appear within a foil's input text (i.e., it may only appear after all the input text for a foil, but before the next foil-start macro); *n* is the position of font *f*; up to four "*n f*" pairs may be specified; the first font named becomes the *prevailing* font; the initial setting is (**H** is a synonym for **G**):

```
.DF 1 H 2 I 3 B 4 S
```

**.DV** [*a*] [*b*] [*c*] [*d*] Alter the vertical spacing between indentation levels; *a* is the spacing for **.A**, *b* is for **.B**, *c* is for **.C**, and *d* is for **.D**; all non-null arguments must be dimensioned; null arguments leave the corresponding spacing unaffected; initial setting is:

```
.DV .5v .5v .5v 0v
```

**.U** *str1* [*str2*] Underline *str1* and concatenate *str2* (if any) to it.

The last four macros in the above list do not cause a break; the **.I** macro causes a break only if it is invoked with more than one argument; all the other macros cause a break.

The macro package also recognizes the following upper-case synonyms for the corresponding lower-case *troff* requests:

```
.AD .BR .CE .FI .HY .NA .NF .NH .NX .SO .SP .TA .TI
```

The **Tm** string produces the trademark symbol.

The input tilde (˜ ) character is translated into a blank on output.

See the user's manual cited below for further details.

FILES

/usr/lib/tmac/tmac.v
/usr/lib/macros/vmca

SEE ALSO

cw(1), eqn(1), mmt(1), tbl(1), troff(1).
*A Macro Package for View Graphs and Slides* by T. A. Dolotta and D. W. Smith.

BUGS

The **.VW** and **.SW** foils are meant to be 9′ ′ wide by 7′ ′ high, but because the typesetter paper is generally only 8′ ′ wide, they are printed 7′ ′ wide by 5.4′ ′ high and have to be enlarged by a factor of 9/7 before use as view graphs; this makes them less than totally useful.

## NAME

pilf, dio – performance improvement in large files and direct I/O

## DESCRIPTION

A PILF file system supports the input or output of large amounts of data with a single physical read or write. This requires special strategies for I/O; when standard I/O operations are applied to a PILF file system, it behaves like a standard 1K file system. A PILF file system is created with the **–P** option of *mkfs*(1M).

A file on a PILF file system is allocated by clusters, each of which is equal in size and consists of contiguous blocks. Performance improvement is seen when the DIO (Direct Input/Output) mechanism is used and no read or write crosses a cluster boundary.

A field in the i-node determines the file's cluster size. A cluster consists of $2^c$ 1K blocks, where *c* is the value in the i-node. The process that creates a PILF file specifies its cluster size using *syslocal*(2); if a process has not yet specified a cluster size, the default cluster size, in the superblock, is used. A file's cluster size is determined when it is created and cannot be changed.

DIO transfers data directly between the process's address space and the disk, bypassing the kernel buffer cache. It is specifically meant to be used on PILF files. DIO is specified with *open* or *fcntl*.

## SEE ALSO

cp(1), mkfs(1M), fsck(1M), fsdb(1M), fcntl(2), fork(2), open(2), syslocal(2), fs(4), inode(4), fcntl(5).

## WARNING

A buffer used for DIO must be on an even address. This is the same degree of alignment as a **short**.

## NAME

prof – profile within a function

## SYNOPSIS

**#define** MARK
**#include <prof.h>**

**void** MARK **(name)**

## DESCRIPTION

*MARK* will introduce a mark called *name* that will be treated the same as a function entry point. Execution of the mark will add to a counter for that mark, and program-counter time spent will be accounted to the immediately preceding mark or to the function if there are no preceding marks within the active function.

*Name* may be any combination of up to six letters, numbers or underscores. Each *name* in a single compilation must be unique, but may be the same as any ordinary program symbol.

For marks to be effective, the symbol MARK must be defined before the header file <*prof.h*> is included. This may be defined by a preprocessor directive as in the synopsis, or by a command line argument, i.e:

```
cc –p –DMARK foo.c
```

If MARK is not defined, the *MARK*(name) statements may be left in the source files containing them and will be ignored.

## EXAMPLE

In this example, marks can be used to determine how much time is spent in each loop. Unless this example is compiled with *MARK* defined on the command line, the marks are ignored.

```
#include <prof.h>

foo( )
{
        int i, j;

        .
        .
        .
        MARK(loop1);
        for (i = 0; i < 2000; i++) {
                . . .
        }
        MARK(loop2);
        for (j = 0; j < 2000; j++) {
                . . .
        }
}
```

## SEE ALSO

prof(1), profil(2), monitor(3C).

## NAME

regexp – regular expression compile and match routines

## SYNOPSIS

**#define INIT** <declarations>
**#define GETC( )** <getc code>
**#define PEEKC( )** <peekc code>
**#define UNGETC(c)** <ungetc code>
**#define RETURN(pointer)** <return code>
**#define ERROR(val)** <error code>

**#include <regexp.h>**

**char *compile (instring, expbuf, endbuf, eof)**
**char *instring, *expbuf, *endbuf;**
**int eof;**

**int step (string, expbuf)**
**char *string, *expbuf;**

**extern char *loc1, *loc2, *locs;**

**extern int circf, sed, nbra;**

## DESCRIPTION

This page describes general-purpose regular expression matching routines in the form of *ed*(1), defined in **/usr/include/regexp.h**. Programs such as *ed*(1), *sed*(1), *grep*(1), *bs*(1), *expr*(1), etc., which perform regular expression matching use this source file. In this way, only this file need be changed to maintain regular expression compatibility.

The interface to this file is unpleasantly complex. Programs that include this file must have the following five macros declared before the "#include <regexp.h>" statement. These macros are used by the *compile* routine.

| | |
|---|---|
| GETC( ) | Return the value of the next character in the regular expression pattern. Successive calls to GETC( ) should return successive characters of the regular expression. |
| PEEKC( ) | Return the next character in the regular expression. Successive calls to PEEKC( ) should return the same character (which should also be the next character returned by GETC( )). |
| UNGETC(*c*) | Cause the argument *c* to be returned by the next call to GETC( ) (and PEEKC( )). No more that one character of pushback is ever needed and this character is guaranteed to be the last character read by GETC( ). The value of the macro UNGETC(*c*) is always ignored. |
| RETURN(*pointer*) | This macro is used on normal exit of the *compile* routine. The value of the argument *pointer* is a pointer to the character after the last character of the compiled regular expression. This is useful to programs which have memory allocation to manage. |
| ERROR(*val*) | This is the abnormal return from the *compile* routine. The argument *val* is an error number (see table below for meanings). This call should never return. |

| ERROR | MEANING |
|---|---|
| 11 | Range endpoint too large. |
| 16 | Bad number. |
| 25 | "\digit" out of range. |
| 36 | Illegal or missing delimiter. |
| 41 | No remembered search string. |
| 42 | \( \) imbalance. |
| 43 | Too many \(. |
| 44 | More than 2 numbers given in \{ \}. |
| 45 | } expected after \. |
| 46 | First number exceeds second in \{ \}. |
| 49 | [ ] imbalance. |
| 50 | Regular expression overflow. |

The syntax of the *compile* routine is as follows:

```
compile(instring, expbuf, endbuf, eof)
```

The first parameter *instring* is never used explicitly by the *compile* routine but is useful for programs that pass down different pointers to input characters. It is sometimes used in the INIT declaration (see below). Programs which call functions to input characters or have characters in an external array can pass down a value of ((char *) 0) for this parameter.

The next parameter *expbuf* is a character pointer. It points to the place where the compiled regular expression will be placed.

The parameter *endbuf* is one more than the highest address where the compiled regular expression may be placed. If the compiled expression cannot fit in (*endbuf–expbuf*) bytes, a call to ERROR(50) is made.

The parameter *eof* is the character which marks the end of the regular expression. For example, in *ed*(1), this character is usually a /.

Each program that includes this file must have a **#define** statement for INIT. This definition will be placed right after the declaration for the function *compile* and the opening curly brace ({). It is used for dependent declarations and initializations. Most often it is used to set a register variable to point the beginning of the regular expression so that this register variable can be used in the declarations for GETC( ), PEEKC( ) and UNGETC( ). Otherwise it can be used to declare external variables that might be used by GETC( ), PEEKC( ) and UNGETC( ). See the example below of the declarations taken from *grep*(1).

There are other functions in this file which perform actual regular expression matching, one of which is the function *step*. The call to *step* is as follows:

```
step(string, expbuf)
```

The first parameter to *step* is a pointer to a string of characters to be checked for a match. This string should be null terminated.

The second parameter *expbuf* is the compiled regular expression which was obtained by a call of the function *compile*.

The function *step* returns non-zero if the given string matches the regular expression, and zero if the expressions do not match. If there is a match, two external character pointers are set as a side effect to the call to *step*. The variable set in *step* is *loc1*. This is a pointer to the first character that matched the regular expression. The variable *loc2*, which is set by the function *advance*, points to the character after the last character that matches the regular expression. Thus if the regular expression matches the entire line, *loc1* will point to the first character of *string* and *loc2* will point to the null at the end of *string*.

*Step* uses the external variable *circf* which is set by *compile* if the regular expression begins with ^. If this is set then *step* will try to match the regular expression to the beginning of the string only. If more than one regular expression is to be compiled before the first is executed the value of *circf* should be saved for each compiled expression and *circf* should be set to that saved value before each call to *step*.

The function *advance* is called from *step* with the same arguments as *step*. The purpose of *step* is to step through the *string* argument and call *advance* until *advance* returns non-zero indicating a match or until the end of *string* is reached. If one wants to constrain *string* to the beginning of the line in all cases, *step* need not be called; simply call *advance*.

When *advance* encounters a * or \{ \} sequence in the regular expression, it will advance its pointer to the string to be matched as far as possible and will recursively call itself trying to match the rest of the string to the rest of the regular expression. As long as there is no match, *advance* will back up along the string until it finds a match or reaches the point in the string that initially matched the * or \{ \}. It is sometimes desirable to stop this backing up before the initial point in the string is reached. If the external character pointer *locs* is equal to the point in the string at sometime during the backing up process, *advance* will break out of the loop that backs up and will return zero. This is used by *ed*(1) and *sed*(1) for substitutions done globally (not just the first occurrence, but the whole line) so, for example, expressions like **s/y*//g** do not loop forever.

The additional external variables *sed* and *nbra* are used for special purposes.

EXAMPLES

The following is an example of how the regular expression macros and calls look from *grep*(1):

```
#define INIT              register char *sp = instring;
#define GETC( )           (*sp++)
#define PEEKC( )          (*sp)
#define UNGETC(c)         (--sp)
#define RETURN(c)         return;
#define ERROR(c)          regerr( )

#include <regexp.h>
...
                (void) compile(*argv, expbuf, &expbuf[ESIZE], '\0' );
...
                if (step(linebuf, expbuf))
                                succeed( );
```

FILES

/usr/include/regexp.h

SEE ALSO

bs(1), ed(1), expr(1), grep(1), sed(1).

BUGS

The handling of *circf* is kludgy.
The actual code is probably easier to understand than this manual page.

## NAME

stat – data returned by stat system call

## SYNOPSIS

**#include <sys/types.h>**
**#include <sys/stat.h>**

## DESCRIPTION

The system calls *stat* and *fstat* return data whose structure is defined by this include file. The encoding of the field *st_mode* is defined in this file also.

```
/*
 * Structure of the result of stat
 */

struct   stat
{
         dev_t    st_dev;
         ino_t    st_ino;
         ushort   st_mode;
         short    st_nlink;
         ushort   st_uid;
         ushort   st_gid;
         dev_t    st_rdev;
         off_t    st_size;
         time_t   st_atime;
         time_t   st_mtime;
         time_t   st_ctime;
};

#define S_IFMT    0170000 /* type of file */
#define S_IFDIR   0040000 /* directory */
#define S_IFCHR   0020000 /* character special */
#define S_IFBLK   0060000 /* block special */
#define S_IFREG   0100000 /* regular */
#define S_IFIFO   0010000 /* fifo */
#define S_ISUID   04000   /* set user id on execution */
#define S_ISGID   02000   /* set group id on execution */
#define S_ISVTX   01000   /* save swapped text even after use */
#define S_IREAD   00400   /* read permission, owner */
#define S_IWRITE  00200   /* write permission, owner */
#define S_IEXEC   00100   /* execute/search permission, owner */
```

## FILES

/usr/include/sys/types.h
/usr/include/sys/stat.h

## SEE ALSO

stat(2), types(5).

## NAME

term – conventional names for terminals

## DESCRIPTION

These names are used by certain commands (e.g., *tabs*(1), *man*(1) and are maintained as part of the shell environment (see *sh*(1), *profile*(4), and *environ*(5)) in the variable $TERM:

| | |
|---|---|
| pt | TM31 |
| freedom | Liberty Freedom 100 |
| 1520 | Datamedia 1520 |
| 1620 | DIABLO 1620 and others using the HyType II printer |
| 1620–12 | same, in 12-pitch mode |
| 2621 | Hewlett-Packard HP2621 series |
| 2631 | Hewlett-Packard 2631 line printer |
| 2631–c | Hewlett-Packard 2631 line printer - compressed mode |
| 2631–e | Hewlett-Packard 2631 line printer - expanded mode |
| 2640 | Hewlett-Packard HP2640 series |
| 2645 | Hewlett-Packard HP264n series (other than the 2640 series) |
| 300 | DASI/DTC/GSI 300 and others using the HyType I printer |
| 300–12 | same, in 12-pitch mode |
| 300s | DASI/DTC/GSI 300s |
| 382 | DTC 382 |
| 300s–12 | same, in 12-pitch mode |
| 3045 | Datamedia 3045 |
| 33 | TELETYPE Model 33 KSR |
| 37 | TELETYPE Model 37 KSR |
| 40–2 | TELETYPE Model 40/2 |
| 40–4 | TELETYPE Model 40/4 |
| 4540 | TELETYPE Model 4540 |
| 3270 | IBM Model 3270 |
| 4000a | Trendata 4000a |
| 4014 | TEKTRONIX 4014 |
| 43 | TELETYPE Model 43 KSR |
| 450 | DASI 450 (same as Diablo 1620) |
| 450–12 | same, in 12-pitch mode |
| 735 | Texas Instruments TI735 and TI725 |
| 745 | Texas Instruments TI745 |
| dumb | generic name for terminals that lack reverse line-feed and other special escape sequences; likely to work when the real terminal type is not known to the program |
| sync | generic name for synchronous TELETYPE 4540-compatible terminals |
| hp | Hewlett-Packard (same as 2645) |
| lp | generic name for a line printer |
| tn1200 | User Electric TermiNet 1200 |
| tn300 | User Electric TermiNet 300 |

Up to 8 characters, chosen from [-a–z0–9], make up a basic terminal name. Terminal sub-models and operational modes are distinguished by suffixes beginning with a –. Names should generally be based on original vendors, rather than local distributors. A terminal acquired from one vendor should not have more than one distinct basic name.

Commands whose behavior depends on the type of terminal should accept arguments of the form –T*term* where *term* is one of the names given above; if no such argument is present, such commands should obtain the terminal type from the environment variable **$TERM**, which, in turn, should contain *term*.

SEE ALSO

man(1), mm(1), nroff(1), sh(1), stty(1), tabs(1), profile(4), environ(5).

BUGS

This is a small candle trying to illuminate a large, dark problem. Programs that ought to adhere to this nomenclature do so somewhat fitfully.

NAME

types – primitive system data types

SYNOPSIS

**#include <sys/types.h>**

DESCRIPTION

The data types defined in the include file are used in operating system code; some data of these types are accessible to user code:

```
typedef struct { int r[1]; } *          physadr;
typedef long            daddr_t;
typedef char *          caddr_t;
typedef unsigned int    uint;
typedef unsigned short  ushort;
typedef ushort          ino_t;
typedef short           cnt_t;
typedef long            time_t;
typedef int             label_t[10];
typedef short           dev_t;
typedef long            off_t;
typedef long            paddr_t;
typedef long            key_t;
```

The form *daddr_t* is used for disk addresses except in an i-node on disk, see *fs*(4). Times are encoded in seconds since 00:00:00 GMT, January 1, 1970. The major and minor parts of a device code specify kind and unit number of a device. Offsets are measured in bytes from the beginning of a file. The *label_t* variables are used to save the processor state while another process is running.

SEE ALSO

fs(4).

## NAME

values – machine-dependent values

## SYNOPSIS

**#include <values.h>**

## DESCRIPTION

This file contains a set of manifest constants, conditionally defined for particular processor architectures.

The model assumed for integers is binary representation (one's or two's complement), where the sign is represented by the value of the high-order bit.

| | |
|---|---|
| BITS(*type* ) | The number of bits in a specified type (e.g., int). |
| HIBITS | The value of a short integer with only the high-order bit set (in most implementations, 0x8000). |
| HIBITL | The value of a long integer with only the high-order bit set (in most implementations, 0x80000000). |
| HIBITI | The value of a regular integer with only the high-order bit set (usually the same as HIBITS or HIBITL). |
| MAXSHORT | The maximum value of a signed short integer (in most implementations, 0x7FFF ≡ 32767). |
| MAXLONG | The maximum value of a signed long integer (in most implementations, 0x7FFFFFFF ≡ 2147483647). |
| MAXINT | The maximum value of a signed regular integer (usually the same as MAXSHORT or MAXLONG). |
| MAXFLOAT, LN_MAXFLOAT | The maximum value of a single-precision floating-point number, and its natural logarithm. |
| MAXDOUBLE, LN_MAXDOUBLE | The maximum value of a double-precision floating-point number, and its natural logarithm. |
| MINFLOAT, LN_MINFLOAT | The minimum positive value of a single-precision floating-point number, and its natural logarithm. |
| MINDOUBLE, LN_MINDOUBLE | The minimum positive value of a double-precision floating-point number, and its natural logarithm. |
| FSIGNIF | The number of significant bits in the mantissa of a single-precision floating-point number. |
| DSIGNIF | The number of significant bits in the mantissa of a double-precision floating-point number. |

## FILES

/usr/include/values.h

## SEE ALSO

intro(3), math(5).

## NAME

varargs – handle variable argument list

## SYNOPSIS

**#include <varargs.h>**

**va_alist**

**va_dcl**

**void va_start(pvar)**
**va_list pvar;**

*type* **va_arg(pvar,** *type***)**
**va_list pvar;**

**void va_end(pvar)**
**va_list pvar;**

## DESCRIPTION

This set of macros allows portable procedures that accept variable argument lists to be written. Routines that have variable argument lists (such as *printf*(3S)) but do not use *varargs* are inherently nonportable, as different machines use different argument-passing conventions.

**va_alist** is used as the parameter list in a function header.

**va_dcl** is a declaration for *va_alist*. No semicolon should follow *va_dcl*.

**va_list** is a type defined for the variable used to traverse the list.

**va_start** is called to initialize *pvar* to the beginning of the list.

**va_arg** will return the next argument in the list pointed to by *pvar*. *Type* is the type the argument is expected to be. Different types can be mixed, but it is up to the routine to know what type of argument is expected, as it cannot be determined at runtime.

**va_end** is used to clean up.

Multiple traversals, each bracketed by *va_start* ... *va_end,* are possible.

## EXAMPLE

This example is a possible implementation of *execl*(2).

```
#include <varargs.h>
#define MAXARGS     100

/*      execl is called by
                execl(file, arg1, arg2, ..., (char *)0);
*/
execl(va_alist)
va_dcl
{
        va_list ap;
        char *file;
        char *args[MAXARGS];
        int argno = 0;

        va_start(ap);
        file == va_arg(ap, char *);
        while ((args[argno++] = va_arg(ap, char *)) != (char *)0)
                ;
        va_end(ap);
        return execv(file, args);
        }
```

SEE ALSO

exec(2), printf(3S).

BUGS

It is up to the calling routine to specify how many arguments there are, since it is not always possible to determine this from the stack frame. For example, *execl* is passed a zero pointer to signal the end of the list. *Printf* can tell how many arguments are there by the format.
It is non-portable to specify a second argument of *char*, *short*, or *float* to *va_arg*, since arguments seen by the called function are not *char*, *short*, or *float*. C converts *char* and *short* arguments to *int* and converts *float* arguments to *double* before passing them to a function.