UNISYS

U 6000 Series System V Programmer's Reference Manual Volume 3

March 1988

Printed in U S America

Priced Item

UP–13713.3

This documentation is based in part on the fourth Berkeley Software Distribution, under license from the Regents of the University of California. We acknowledge the following individuals and institutions for their role in its development:

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

6000/50

Customization Package

This customization package contains changes to your *Programmer's Reference Manual, Volume 3,* which reflect the *AT&T System V Release 3.1* and the value-added features of the *Unisys System V Operating System.* Please add these pages to your base manual to produce a fully customized *Programmer's Reference Manual.*

To fully customize your Programmer's Reference Manual, Volume 1, replace the existing generic manual cover with the new customized manual cover.

Customization Directions

The table below indicates the name of the document and directions for making your manual current. The document found in your customization package may be used to replace an already existing document of the same name, or it may be added as a new document to the manual. You may also be directed to remove an existing document from the current manual. For the location of specific documents, please refer to the Table of Contents.

New Customization Document	*Customization Directions*
Table of Contents	replace old with new
intro(4)	replace old with new
a.out(4)	replace old with new
cprofile(4)	add new
errfile(4)	add new
filehdr(4)	replace old with new
gettydefs(4)	replace old with new
inittab(4)	replace old with new
limits(4)	replace old with new
master(4)	add new
otermcap(4)	add new
rfmaster(4)	replace old with new
system(4)	replace old with new
ttytype(4)	add new
tz(4)	add new
eqnchar(5)	add new
man(5)	add new
me(5)	add new
mm(5)	add new
mptx(5)	add new
ms(5)	add new
mvt(5)	add new

Table of Contents

(The following are contained in three volumes.)

NAME

intro - introduction to file formats

DESCRIPTION

This section outlines the formats of various files. The C structure declarations for the file formats are given where applicable. Usually, the header files containing these structure declarations can be found in the directories **/usr/include** or **/usr/include/sys**. For inclusion in C language programs, however, the syntax **#include <filename.h>** or **#include <sys/filename.h>** should be used.

Entries suffixed by **(4B)** describe the configuration files used with the System V Berkeley networking package (BSD Package). These files can be manipulated directly (using a text editor) or with *netman*(1BM).

[This page left blank.]

NAME

a.out - common assembler and link editor output

SYNOPSIS

#include <a.out.h>

DESCRIPTION

The file name **a.out** is the default output file name from the link editor *ld*(1). The link editor will make *a.out* executable if there were no errors in linking. The output file of the assembler *as*(1), also follows the common object file format of the *a.out* file although the default file name is different.

A common object file consists of a file header, a System V system header (if the file is link editor output), a table of section headers, relocation information, (optional) line numbers, a symbol table, and a string table. The order is given below.

File header.
System V system header.
Section 1 header.
...
Section n header.
Section 1 data.
...
Section n data.
Section 1 relocation.
...
Section n relocation.
Section 1 line numbers.
...
Section n line numbers.
Symbol table.
String table.

The last three parts of an object file (line numbers, symbol table, and string table) may be missing if the program was linked with the **-s** option of *ld*(1) or if they were removed by *strip*(1). Also note that the relocation information will be absent after linking unless the **-r** option of *ld*(1) was used. The string table exists only if the symbol table contains symbols with names longer than eight characters.

The sizes of each section (contained in the header, discussed below) are in bytes.

When an **a.out** file is loaded into memory for execution, three logical segments are set up: the text segment, the data segment (initialized data followed by uninitialized, the latter actually being initialized to all 0's), and a stack. On the the text segment starts at location 0x00000000.

The **a.out** file produced by *ld*(1) may have one of two magic numbers in the first field of the System V system header. A magic number of 0410 indicates that the executable must be swapped through the private swapping store of the System V system, while the magic number 0413 causes the system to attempt to page the text directly from the **a.out** file.

For 0410 executable files, the text section is loaded at virtual location 0x00000000. The data section is loaded immediately following the end of the text section.

For 0413 executable files, the headers (file header, System V system header, and section headers) are loaded at the beginning of the text segment and the text immediately follows the headers in the user address space. The first text address will equal the sum of the sizes of the headers, and will vary depending on the number of sections in the **a.out** file. In an **a.out** file with 3 sections (.text, .data, and .bss) the first text address is at 0x000000d0. The data section starts in the next page table directory after the last one used by the text section, in the first page of that directory, with an offset into that page equal to the page offset of the data section in the **a.out** file. That is to say, given that *etext* is the address of the last byte of the text section, and *dataoffset* is the offset of the data section in the **a.out** file, the first byte of the data section will be at ((etext + 0x3FFFFF) & 0xFFC00000) + (dataoffset & 0xFFF).

Regardless of the magic number, the stack begins at address 0xbfffffff and grows to lower memory locations. The stack is automatically extended as required. The data segment is extended only as requested by the *brk*(2) system call.

For relocatable files the value of a word in the text or data portions that is not a reference to an undefined external symbol is exactly the value that will appear in memory when the file is executed. If a word in the text involves a reference to an undefined external symbol, there will be a relocation entry for the word, the storage class of the symbol-table entry for the symbol will be marked as an "external symbol", and the value and section number of the symbol-table entry will be undefined. When the file is processed by the link editor and the external symbol becomes defined, the value of the symbol will be added to the word in the file.

File Header

The format of the **filehdr** header is:

```
struct filehdr
{
        unsigned short  f_magic;   /* magic number */
        unsigned short  f_nscns;   /* number of sections */
        long            f_timdat;  /* time and date stamp */
        long            f_symptr;  /* file ptr to symtab */
        long            f_nsyms;   /* # symtab entries */
        unsigned short  f_opthdr;  /* sizeof(opt hdr) */
        unsigned short  f_flags;   /* flags */
};
```

System V Header

The format of the System V system header is:

```
typedef struct aouthdr
{
        short magic;       /* magic number */
        short vstamp;      /* version stamp */
        long  tsize;       /* text size in bytes, padded */
        long  dsize;       /* initialized data (.data) */
        long  bsize;       /* uninitialized data (.bss) */
        long  entry;       /* entry point */
        long  text_start;  /* base of text used for this file */
        long  data_start;  /* base of data used for this file */
} AOUTHDR;
```

Section Header

The format of the section header is:

```
struct scnhdr
{
      char           s_name[SYMNMLEN];/* section name */
      long           s_paddr;   /* physical address */
      long           s_vaddr;   /* virtual address */
      long           s_size;    /* section size */
      long           s_scnptr;  /* file ptr to raw data */
      long           s_relptr;  /* file ptr to relocation */
      long           s_lnnoptr; /* file ptr to line numbers */
      unsigned short s_nreloc;  /* # reloc entries */
      unsigned short s_nlnno;   /* # line number entries */
      long           s_flags;   /* flags */
};
```

Relocation

Object files have one relocation entry for each relocatable reference in the text or data. If relocation information is present, it will be in the following format:

```
struct reloc
{
      long   r_vaddr;   /* (virtual) address of reference */
      long   r_symndx;  /* index into symbol table */
      ushort r_type;    /* relocation type */
};
```

The startof the relocationinformation is*s_relptr* from the section header. If there is no relocation information, *s_relptr* is 0.

Symbol Table

The format of each symbol in the symbol table is:

```
#define  SYMNMLEN  8
#define  FILNMLEN  14
#define  DIMNUM    4

struct syment
{
    union                        /* all ways to get a symbol */
                                 /* name */
    {
        char _n_name[SYMNMLEN]; /* name of symbol */
```

```
            struct
            {
                long    _n_zeroes;  /* == 0L if in string table */
                long    _n_offset;  /* location in string table */
            } _n_n;
            char *_n_nptr[2];       /* allows overlaying */
        } _n;
        long     n_value;           /* value of symbol */
        short            n_scnum;   /* section number */
        unsigned short   n_type;    /* type and derived type */
        char     n_sclass;          /* storage class */
        char     n_numaux;          /* number of aux entries */
};

#define  n_name    _n._n_name
#define  n_zeroes  _n._n_n._n_zeroes
#define  n_offset  _n._n_n._n_offset
#define  n_nptr    _n._n_nptr[1]
```

Some symbols require more information than a single entry; they are followed by *auxiliary entries* that are the same size as a symbol entry. The format follows.

```
union auxent {
    struct {
        long    x_tagndx;
        union {
                struct {
                        unsigned short  x_lnno;
                        unsigned short  x_size;
                } x_lnsz;
                long    x_fsize;
        } x_misc;
        union {
                struct {
                        long    x_lnnoptr;
                        long    x_endndx;
                }  x_fcn;
                struct {
                        unsigned short  x_dimen[DIMNUM];
                } x_ary;
        } x_fcnary;
        unsigned short  x_tvndx;
        } x_sym;

        struct {
            char     x_fname[FILNMLEN];
        } x_file;

        struct {
            long           x_scnlen;
            unsigned short x_nreloc;
            unsigned short x_nlinno;
        } x_scn;

        struct {
            long           x_tvfill;
            unsigned short x_tvlen;
            unsigned short x_tvran[2];
        } x_tv;
    };
```

Indexes of symbol table entries begin at *zero*. The start of the symbol table is *f_symptr* (from the file header) bytes from the beginning of the file. If the symbol table is stripped, *f_symptr* is 0. The string table (if one exists) begins at *f_symptr* + (*f_nsyms* * SYMESZ) bytes from the beginning of the file.

SEE ALSO

as(1), cc(1), ld(1), brk(2), filehdr(4), ldfcn(4), linenum(4), reloc(4), scnhdr(4), syms(4).

NAME

cprofile - setting up a C shell environment at login time

DESCRIPTION

Cprofile is for use with *csh*(1). For every user of *csh* the system file **/etc/cprofile** is executed immediately upon login. If the user's login directory contains a file named **.cshrc**, that file will then be executed, followed by commands from the **.login** file.

The following example is typical for a user's **.cshrc** file:

```
setenv PATH :$PATH:$HOME/bin
setenv MAIL /usr/mail/myname
setenv TERM pt
umask 022
```

The system file **/etc/cprofile** can be customized to set the **TERM** environment variable via *tset*(1).

For further information about setting variables, see *csh*(1) and *sh*(1).

FILES

/.login
/.cshrc
/.logout
/etc/cprofile

SEE ALSO

ttytype(4), environ(5), term(5).
csh(1), env(1), login(1), mail(1), sh(1), stty(1), tset(1) in the *User's Reference Manual*.

[This page left blank.]

NAME

errfile - error-log file format

SYNOPSIS

#include <sys/erec.h>

DESCRIPTION

When hardware errors are detected by the system, an error record is generated and passed to the error-logging daemon for recording in the error log for later analysis. The default error log is **/usr/adm/errfile**.

The format of an error record depends on the type of error that was encountered. Every record, however, has a header with the following format:

```
struct errhdr {
        short   e_type;  /* record type */
        short   e_len;   /* bytes in record (inc hdr) */
        time_t  e_time;  /* time of day */
};
```

The permissible record types are as follows:

```
#define  E_GOTS     010  /* start */
#define E_STOP      012  /* stop */
#define E_TCHG      013  /* time change */
#define E_CCHG      014  /* configuration change */
#define E_BLK       020  /* block device error */
#define E_STRAY     030  /* stray interrupt */
#define E_PRTY      031  /* memory parity */
#define E_CONS      040  /* console string */
#define E_CONR      041  /* console record */
#define E_CONO      042  /* console overflow */
#define E_SERIAL    043  /* serial device driver error */
```

Some records in the error file are of an administrative nature. These include the startup record that is entered into the file when logging is activated, the stop record that is written if the daemon is terminated "gracefully," and the time-change record that is used to account for changes in the system's time-of-day. These records have the following formats:

```
struct estart {
        struct utsname  e_name;     /* system names */
        long            e_syssize;  /* system memory size */
        short           e_cpu;      /* CPU type */
        short           e_bconf;    /* block device */
                                    /* configuration */
        char            e_panic;    /* if reboot from panic, */
                                    /* what was it */
};

#define eend errhdr                 /* record header */

struct etimchg {
        time_t e_ntime;             /* new time */
};
```

Stray interrupts cause a record with the following format to be logged:

```
struct estray {
        physadr   e_saddr;   /* stray loc or device addr */
        short     e_sbacty;  /* active block devices */
};
```

Memory subsystem error causes the following record to be generated:

```
struct eparity {
        int  e_status  /* parity status register */
        int  e_addr    /* address of parity error */
                       /* (not supported on the */
                       /* ) */
};
```

Note that there is no hardware support to provide the address of a parity error on the .

Error records for block devices have the following format:

```
struct eblock {
        struct iostat {
           long     io_ops;     /* number read/writes */
           long     io_misc;    /* number "other" operations */
           ushort   io_unlog;   /* number unlogged errors */
        }           e_stats;
        paddr_t     e_memadd;   /* buffer memory address */
        daddr_t     e_bnum;     /* logical block number */
        int         e_bytes;    /* number bytes to transfer */
        int         e_bacty;    /* other block I/O activity */
        short       e_trkoff;   /* logical device start track */
        ushort      e_rtry;     /* number retries */
        short       e_trks      /* number of heads */
        short       e_secs      /* number of physical sectors */
                                /* per track */
        short       e_ctlr      /* controller type */
        short       e_major     /* major device number */
        short       e_minor     /* minor device number */
        short       e_bflags;   /* read/write, error, and so on */
};
```

The following values are used in the *e_bflags* word:

```
#define E_WRITE   0    /* write operation */
#define E_READ    1    /* read operation */
#define E_NOIO    02   /* no I/O pending */
#define E_PHYS    04   /* physical I/O */
#define E_MAP     010  /* Unibus map in use */
#define E_ERROR   020  /* I/O failed */
```

The error types CONS and CONO are flagged by *errdemon*(1M) and *errdead* and written to the console log **/etc/log/confile**.

A serial driver error generates the following reports:

```
struct eserial {
        ushort  e_type  /* type of error */
        ushort  e_dev   /* which physical port */
};
```

The following types exist for e_type:

```
#define ECHLOS     0x1  /* character lost in input FIFO */
#define ERXORUN    0x2  /* receiver overrun */
#define ENOCLIST   0x4  /* no new clist available */
#define ENORBUF    0x8  /* no receive buffer available */
```

A configuration change (loading or unloading a loadable driver) generates the following report:

```
struct econfchg {
        short  e_devnum;  /* major device number */
        char   e_cflag;   /* 1 = attach, 0 = detach */
        char   e_dtype;   /* block or char device */
};
```

SEE ALSO

errdemon(1M).

NAME

filehdr - file header for common object files

SYNOPSIS

#include <filehdr.h>

DESCRIPTION

Every common object file begins with a 20-byte header. The following C **struct** declaration is used:

```
struct        filehdr
{
      unsigned short  f_magic ;   /* magic number */
      unsigned short  f_nscns ;   /* number of sections */
      long            f_timdat ;  /* time & date stamp */
      long            f_symptr ;  /* file ptr to symtab */
      long            f_nsyms ;   /* # symtab entries */
      unsigned short  f_opthdr ;  /* sizeof(opt hdr) */
      unsigned short  f_flags ;   /* flags */
} ;
```

F_symptr is the byte offset into the file at which the symbol table can be found. Its value can be used as the offset in *fseek*(3S) to position an I/O stream to the symbol table. The System V system optional header is 28-bytes. The valid magic numbers are given below:

```
#define  I286SMAGIC  0512  /* Intel 286 computers, */
                           /* small model programs */
#define  I386MAGIC   0514  /* Intel 386 computers */
                           /* () */
#define  I286LMAGIC  0522  /* Intel 286 computers, */
                           /* large model programs */
```

The value in*f_timdat* is obtained from the *time*(2) system call. Flag bits currently defined are:

```
#define  F_RELFLG   0000001 /* relocation entries stripped */
#define  F_EXEC     0000002  /* file is executable */
#define  F_LNNO     0000004  /* line numbers stripped */
#define  F_LSYMS    0000010  /* local symbols stripped */
#define  F_MINMAL   0000020  /* minimal object file */
#define  F_UPDATE   0000040  /* update file, ogen produced */
#define  F_SWABD    0000100  /* file is "pre-swabbed" */
#define  F_AR16WR   0000200  /* 16-bit DEC host */
```

```
#define  F_AR32WR   0000400  /* 386, 32-bit DEC host */
#define  F_AR32W    0001000  /* non-DEC host */
#define  F_PATCH    0002000  /* "patch" list in opt hdr */
#define  F_80186    010000   /* contains 80186 instructions */
#define  F_80286    020000   /* contains 80286 instructions */
```

SEE ALSO

time(2), fseek(3S), a.out(4).

NAME

gettydefs - speed and terminal settings used by getty

DESCRIPTION

The **/etc/gettydefs** file contains information used by *getty*(1M) to set up the speed and terminal settings for a line. It supplies information on what the *login* prompt should look like. It also supplies the speed to try next if the user indicates the current speed is not correct by typing a <*break*> character.

Each entry in **/etc/gettydefs** has the following format:

```
label# initial-flags # final-flags # login-prompt #next-label
```

Each entry is followed by a blank line. The various fields can contain quoted characters of the form **b**, **n**, **c**, and so forth, as well as *nnn*, where *nnn* is the octal value of the desired character. The various fields are:

label	This is the string against which *getty* tries to match its second argument. It is often the speed, such as *1200*, at which the terminal is supposed to run, but it need not be (see below).
initial-flags	These flags are the initial *ioctl*(2) settings to which the terminal is to be set if a terminal type is not specified to *getty*. The flags that *getty* understands are the same as the ones listed in **/usr/include/sys/termio.h** [see *termio*(7)]. Normally only the speed flag is required in the *initial-flags*. *Getty* automatically sets the terminal to raw input mode and takes care of most of the other flags. The *initial-flag* settings remain in effect until *getty* executes *login*(1).
final-flags	These flags take the same values as the *initial-flags* and are set just before *getty* executes *login*. The speed flag is again required. The composite flag **SANE** takes care of most of the other flags that need to be set so that the processor and terminal are communicating in a rational fashion. The other two commonly specified *final-flags* are **TAB3**, so that tabs are sent to the terminal as spaces, and **HUPCL**, so that the line is hung up on the final close.

login-prompt	This entire field is printed as the *login-prompt*. Unlike the above fields where white space is ignored (a space, tab, or new-line), they are included in the *login-prompt* field.
next-label	If this entry does not specify the desired speed, indicated by the user typing a <*break*> character, then *getty* will search for the entry with *next-label* as its *label* field and set up the terminal for those settings. Usually, a series of speeds are linked together in this fashion, into a closed set; for instance, **2400** linked to **1200**, which in turn is linked to **300**, which finally is linked to **2400**.

If *getty* is called without a second argument, then the first entry of **/etc/gettydefs** is used, thus making the first entry of **/etc/gettydefs** the default entry. It is also used if *getty* can not find the specified *label*. If **/etc/gettydefs** itself is missing, there is one entry built into the command which will bring up a terminal at **9600** baud.

It is strongly recommended that after making or modifying **/etc/gettydefs**, it be run through *getty* with the check option to be sure there are no errors.

FILES

/etc/gettydefs

SEE ALSO

ioctl(2).
getty(1M), termio(7) in the *Administrator's Reference Manual*.
login(1) in the *User's Reference Manual*.

NAME

inittab - script for the init process

DESCRIPTION

The **inittab** file supplies the script to *init*'s role as a general process dispatcher. The process that constitutes the majority of *init*'s process dispatching activities is the line process **/etc/getty** that initiates individual terminal lines. Other processes typically dispatched by *init* are daemons and the shell.

The **inittab** file is composed of entries that are position dependent and have the following format:

id:rstate:action:process

Each entry is delimited by a newline; however, a backslash (\) preceding a newline indicates a continuation of the entry. Up to 512 characters per entry are permitted. Comments may be inserted in the *process* field using the *sh*(1) convention for comments. Comments for lines that spawn *getty*s are displayed by the *who*(1) command. It is expected that they will contain some information about the line such as the location. There are no limits (other than maximum entry size) imposed on the number of entries within the **inittab** file. The entry fields are:

id This is one or two characters used to uniquely identify an entry.

rstate This defines the *run-level* in which this entry is to be processed. *Run-levels* effectively correspond to a configuration of processes in the system. That is, each process spawned by *init* is assigned a *run-level* or *run-levels* in which it is allowed to exist. The *run-levels* are represented by a number ranging from **0** through **6**. As an example, if the system is in *run-level* **1**, only those entries having a **1** in the *rstate* field will be processed. When *init* is requested to change *run-levels,* all processes which do not have an entry in the *rstate* field for the target *run-level* will be sent the warning signal (**SIGTERM**) and allowed a 20-second grace period before being forcibly terminated by a kill signal (**SIGKILL**). The *rstate* field can define multiple *run-levels* for a process by selecting more than one *run-level* in any combination from **0-6**. If no *run-level* is specified, then the process is assumed to be valid at all *run-levels* **0-6**. There are three other values, **a**, **b** and **c**, which can appear in the *rstate* field, even though they are not true *run-levels*. Entries which have these characters in the *rstate* field are processed only when the *telinit* [see *init*(1M)] process requests them to be run (regardless of the current *run-level* of the system). They differ from *run-levels* in that *init* can never enter *run-level* **a**, **b** or **c**. Also, a request for the execution of any of these processes does not change the current *run-level*. Furthermore, a process started by an **a**, **b** or **c** command is not killed when *init* changes levels. They are only killed if their line in **/etc/inittab** is marked **off** in the *action* field, their line is deleted entirely from **/etc/inittab**, or *init* goes into the *SINGLE USER* state.

action Key words in this field tell *init* how to treat the process specified in the *process* field. The actions recognized by *init* are as follows:

respawn If the process does not exist then start the process, do not wait for its termination (continue scanning the **inittab** file), and when it dies restart the process. If the process currently exists then do nothing and continue scanning the **inittab** file.

wait Upon *init*'s entering the *run-level* that matches the entry's *rstate*, start the process and wait for its termination. All subsequent reads of the **inittab** file while *init* is in the same *run-level* will cause *init* to ignore this entry.

once Upon *init*'s entering a *run-level* that matches the entry's *rstate*, start the process, do not wait for its termination. When it dies, do not restart the process. If upon entering a new *run-level*, where the process is still running from a

	previous *run-level* change, the program will not be restarted.
boot	The entry is to be processed only at *init*'s boot-time read of the **inittab** file. *Init* is to start the process, not wait for its termination; and when it dies, not restart the process. In order for this instruction to be meaningful, the *rstate* should be the default or it must match *init*'s *run-level* at boot time. This action is useful for an initialization function following a hardware reboot of the system.
bootwait	The entry is to be processed only at *init*'s boot-time read of the **inittab** file. *Init* is to start the process, wait for its termination, and when it dies, not restart the process.
powerfail	Execute the process associated with this entry only when *init* receives a power fail signal [**SIGPWR** see *signal*(2)].
powerwait	Execute the process associated with this entry only when *init* receives a power fail signal (**SIGPWR**) and wait until it terminates before continuing any processing of **inittab**.
off	If the process associated with this entry is currently running, send the warning signal (**SIGTERM**) and wait 20 seconds before forcibly terminating the process

via the kill signal (**SIGKILL**). If the process is nonexistent, ignore the entry.

ondemand This instruction is really a synonym for the **respawn** action. It is functionally identical to **respawn** but is given a different keyword in order to divorce its association with *run-levels*. This is used only with the **a**, **b** or **c** values described in the *rstate* field.

initdefault An entry with this *action* is only scanned when *init* is initially invoked. *Init* uses this entry, if it exists, to determine which *run-level* to enter initially. It does this by taking the highest *run-level* specified in the **rstate** field and using that as its initial state. If the *rstate* field is empty, this is interpreted as **0123456** and so *init* will enter *run-level* **6**. Additionally, if *init* does not find an **initdefault** entry in **/etc/inittab**, then it will request an initial *run-level* from the user at reboot time.

sysinit Entries of this type are executed before *init* tries to access the console. It is expected that this entry will be only used to initialize devices on which *init* might try to ask the *run-level* question. These entries are executed and waited for before continuing.

process This is a *sh* command to be executed. The entire **process** field is prefixed with *exec* and passed to a forked *sh* as **sh -c 'exec** *command*'. For this reason, any legal *sh* syntax can appear in the *process* field. Comments can be inserted with the ; *#comment* syntax.

FILES

/etc/inittab

SEE ALSO

exec(2), open(2), signal(2).

getty(1M), init(1M) in the *Administrator's Reference Manual*.
sh(1), who(1) in the *User's Reference Manual*.

[This page left blank.]

NAME

limits - file header for implementation-specific constants

SYNOPSIS

#include <limits.h>

DESCRIPTION

The header file **<limits.h>** is a list of magnitude limitations imposed by a specific implementation of the operating system. All values are specified in decimal.

```
#define  ARG_MAX    5120    /* max length of arguments to exec */
#define  CHAR_BIT   8       /* # of bits in a "char" */
#define  CHAR_MAX   127     /* max integer value of a "char" */
#define  CHAR_MIN   -128    /* min integer value of a "char" */
#define  CHILD_MAX  25      /* max # of processes per user ID */
#define  CLK_TCK    100     /* # of clock ticks per second */
#define  DBL_DIG    16      /* digits of precision of a "double" */
#define  DBL_MAX    1.79769313486231470e+308  /* max decimal value */
                                              /* of a "double" */
#define  DBL_MIN    4.94065645841246544e-324  /* min decimal value */
                                              /* of a "double" */
#define  FCHR_MAX   1048576      /* max size of a file in bytes */
#define  FLT_DIG    7       /* digits of precision of a "float" */
#define  FLT_MAX    3.40282346638528860e+38   /* max decimal value */
                                              /* of a "float" */
#define  FLT_MIN    1.40129846432481707e-45   /* min decimal value*/
                                              /*  of a "float" */
#define  HUGE_VAL   3.40282346638528860e+38   /* error value */
                                              /* returned by */
                                              /* Math lib */
#define  INT_MAX    2147483647   /* max decimal value of an "int" */
#define  INT_MIN    -2147483648  /* min decimal value of an "int" */
#define  LINK_MAX   32767   /* max # of links to a single file */
#define  LONG_MAX   2147483647   /* max decimal value of a "long" */
#define  LONG_MIN   -2147483648  /* min decimal value of a "long" */
#define  NAME_MAX   14      /* max # of characters in a file name */
#define  OPEN_MAX   20      /* max # of files a process can have */
                            /* open */
#define  PASS_MAX   8       /* max # of characters in a password */
#define  PATH_MAX   256     /* max # of characters in a path name */
#define  PID_MAX    30000   /* max value for a process ID */
#define  PIPE_BUF   5120    /* max # bytes atomic in write to a */
```

```
                                /* pipe */
#define  PIPE_MAX   5120    /* max # bytes written to a pipe in a */
                                /* write */
#define  SHRT_MAX   32767   /* max decimal value of a "short" */
#define  SHRT_MIN   -32767 /* min decimal value of a "short" */
#define  STD_BLK    1024    /* # bytes in a physical I/O block */
#define  SYS_NMLN   9       /* # of chars in uname-returned strings */
#define  UID_MAX    30000   /* max value for a user or group ID */
#define  USI_MAX    4294967296    /* max decimal value of an */
                                      /* "unsigned" */
#define  WORD_BIT   32      /* # of bits in a "word" or "int" */
```

NAME

master - master device information table

DESCRIPTION

This file is used by the *config*(1M) program to obtain device information that enables it to generate the configuration files. Do *not* modify it unless you *fully* understand its construction. The file consists of 3 parts, each separated by a line with a dollar sign ($) in column 1. Part 1 contains device information; part 2 contains names of devices that have aliases; part 3 contains tunable parameter information. Any line with an asterisk (*) in column 1 is treated as a comment.

Part 1 contains lines consisting of between 6 and 9 fields, with the fields delimited by tabs and/or blanks:

Field 1: device name (32 characters maximum).

Field 2: device mask. Each letter indicates that the handler exists:

e	has release handler for downloadable drivers.
t	tty header exists.
n	initialization handler.
o	open handler.
c	close handler.
r	read handler.
w	write handler.
i	ioctl handler.
b	bioctl handler.
s	strategy handler
a	start handler
p	print handler
f	info structure (stream driver only)
u	input handler (line discipline only)
d	output handler (line discipline only)
m	modem interrupt handler (line discipline only)

Field 3: device type indicator:

r	required device.
b	block device.
c	character device.
l	line discipline
m	stream module

	s	software module
Field 4:	handler prefix (4 characters maximum).	
Field 5:	major device number for block-type device.	
Field 6:	major device number for character-type device. For line disciplines, this is the slot number in the linesw table where the line discipline is installed. For software modules it is a unique number used to identify the module. This field is not used for stream modules.	
Fields 7-9:	up to three additional decimal numbers may be specified. These values are passed to the driver's init routine.	

Part 2 contains lines with 2 fields each:

Field 1:	alias name of device (32 characters maximum).
Field 2:	reference name of device (32 characters maximum; specified in part 1).

Part 3 contains lines with 3 fields each:

Field 1:	parameter name (as it appears in description file; 32 characters maximum).
Field 2:	parameter name (as it will appear in the **config.h** file; 32 characters maximum).
Field 3:	default parameter value (32 characters maximum).

EXAMPLE

Below is an abbreviated example of a master file:

```
gdfp        ocsrwpibn bc    gd      14      14
mem         rw        cr    mm      0       18
sxtl        ud        1     sxt     0       1
timod       f         m     tim     0       0
tramdisk    ne        s     ramd    0       6       7
$
floppy      gdfp
$
tablesize   TABLESIZE 100
xlimit      XYZ_LIMIT 127
```

This abbreviated master filedescribes the following:

- A driver called gdfp, which has open, close, strategy, read, write, print, ioctl, bioctl, and init routines. The driver prefix is "gd," so the routines are called gdopen, gdclose, gdstrategy, and so forth. It is both a block and a character device, with both the block and the char major numbers equal to 14.
- A character device driver called mem, with read and write routines (mmread/mmwrite). The open and close routines default to a system-supplied null routine. Mem is specified to be a required device, so it must appear in the description file. The character major number is 18.
- A line discipline called sxtl with just an input and output routine (sxtin/sxtout). It is installed in slot 1 in the linesw table.
- A stream module called timod, with just an info structure (timinfo).
- A software module called tramdisk, with just an init and release routine (ramdinit/ramdrelease). The unique identifying number is 6. The init routine is called with an argument of 7.
- An alias "floppy," which is defined to be identical to "gdfp."
- Two tunable parameters, named tablesize and xlimit, which will generate the defined constants TABLESIZE and XYZ_LIMIT in the **config.h** file. If their values are not specified in the description file, they default to 100 and 128 respectively.

FILES

/etc/master

SEE ALSO

config(1M) in the *Administrator's Reference Manual*.

[This page left blank.]

NAME

otermcap - terminal capability data base

SYNOPSIS

/etc/termcap

DESCRIPTION

This entry describes terminal-independent programming conventions that originate at UC Berkeley. This support is provided for compatibility with earlier software versions; these conventions will no longer be supported in the future. UNIX System V initially borrowed *termcap* but has since changed to the *terminfo*(4) convention. System V continues to support *termcap* so as to be compatible with the Berkeley version of the UNIX System. But use *terminfo* in new programs.

Termcap programs work from information supplied through the **TERM** and **TERMCAP** environment variables. The location of the description depends on the value of **TERMCAP**:

- If **TERMCAP** is not set or is empty, **TERM** is the name of an description in **/etc/termcap**.
- If **TERMCAP** has a value that begins with a /, **TERM** is the name of an description in the file named by **TERMCAP**.
- If **TERMCAP** begins with any character except /, **TERMCAP** contains the description.

A description begins with a list of its names, separated by vertical bars. The rest of the description is a list of capabilities, separated by colons. If you use more than one line, precede each newline except the last with :\. Here's a simple example.

```
d5|vt50|dec vt50:\
    :bs:cd=\EJ:ce=\EK:cl=\EH\EJ:co#80:li#12:\
    :nd=\EC:pt:up=\EA:
```

There are three kinds of capabilities:

- *Boolean*. These indicate the presence or absence of a terminal feature by their presence or absence. Boolean capabilities consist of two characters (the capability name).
- *Numeric*. These indicate some numeric value for the terminal, such as screen size or delay required by a standard character. Numeric capabilities consist of two characters (the capability name), followed by a #, followed by a decimal number.
- *String*. These indicate a sequence that is performs some operation on the terminal. String capabilities consist of two characters (the capability name), optionally followed by a delay, followed by a string.

  The delay is the number of milliseconds the program must wait after using the sequence; specify no more than one decimal place. If the delay is proportional to the number of lines affected, end it with a *.

  The string is a sequence of characters. The following subsequences are specially interpreted.

\E	Escape Character
^x	Control-x
\n	Newline
\r	Return
\t	Tab
\b	Backspace
\f	Formfeed
\xxx	Octal value of xxx
\072	: in string
\200	null (\000 doesn't work)

  Octal numbers must be three digits long.

  Some strings are interpreted further, such as **cm**. See "Cursor Addresses and Other Variables," below.

You can follow any capability name with an @, to indicate that the terminal lacks the capability. This is only useful in conjunction with the **tc** capability; see "Similar Terminals," below.

Here is a list of standard capabilities. (P) indicates a string that might require padding; (P*) indicates a string that might require proportional padding.

Name	Type	Pad?	Description
ae	str	(P)	Ends alternate character set.
al	str	(P*)	Adds new blank line.
am	bool		Terminal has automatic margins.
as	str	(P)	Starts alternate character set.
bc	str		Backspace if not control-h.
bs	bool		Terminal can backspace with control-h.
bt	str	(P)	Back tab.
bw	bool		Backspace wraps from column 0 to last column.
CC	str		Command character in prototype if terminal settable.
cd	str	(P*)	Clears to end of display.
ce	str	(P)	Clears to end of line.
ch	str	(P)	Moves cursor horizontally to specified column.
cl	str	(P*)	Clears screen.
cm	str	(P)	Moves cursor to specified row and column.
co	num		Number of columns in a line.
cr	str	(P*)	Carriage return if not control-m.
cs	str	(P)	Change scrolling region.
cv	str	(P)	Moves cursor vertically to specified row.
da	bool		Display can be retained above.
dB	num		Delay after backspace, in milliseconds.
db	bool		Display can be retained below.
dC	num		Delay after carriage return, in milliseconds.

Name	Type	Pad?	Description
dc	str	(P*)	Delete character.
dF	num		Delay after form feed, in milliseconds.
dl	str	(P*)	Deletes line.
dm	str		Enters delete mode.
dN	num		Delay after newline, in milliseconds.
do	str		Goes down one line.
dT	num		Delay after tab, in milliseconds.
ed	str		Ends delete mode.
ei	str		Ends insert mode; give an empty string if you've defined **ic**.
eo	str		Can erase overstrikes with a blank.
ff	str	(P*)	Hardcopy terminal page eject if not form feed.
hc	bool		Hardcopy terminal.
hd	str		Half-line down (forward 1/2 linefeed).
ho	str		Move cursor to upper left corner (home).
hu	str		Half-line up (reverse 1/2 linefeed).
hz	str		Hazeltine or other terminal that can't print ~'s.
ic	str	(P)	Insert character.
if	str		Name of file containing terminal initialization.
im	bool		Starts insert mode; give an empty string if you've defined **ic**.
in	bool		Insert mode distinguishes nulls on display.
ip	str	(P*)	Pad after insertion.
is	str		Terminal initialization.
k0-k9	str		Sent by special (usually numeric) function keys. If programmable, set with **is**, **if**, **vs**, or **ti**.
kb	str		Sent by backspace key.
kd	str		Sent by terminal down arrow key.
ke	str		Ends keypad transmit mode.

Name	Type	Pad?	Description
kh	str		Sent by home key.
kl	str		Sent by terminal left arrow key.
kn	num		Number of special function keys.
ko	str		Terminal capabilities that have keys.
kr	str		Sent by terminal right arrow key.
ks	str		Begin keypad transmit mode.
ku	str		Sent by terminal up arrow key.
l0-l9	str		Labels on special function keys.
li	num		Number of lines on screen or page.
ll	str		Last line, first column.
ma	str		Command key map; used by *ex* version 2 (System V uses version 3).
mi	bool		Safe to move while in insert mode.
ml	str		Memory lock on above cursor.
ms	bool		Safe to move while in standout and underline mode.
mu	str		Memory unlock (turn off memory lock).
nc	bool		No correctly working carriage return (DM2500,H2000).
nd	str		Non-destructive space (cursor right).
nl	str	(P*)	Begin a new line if not newline.
ns	bool		A video terminal that doesn't scroll.
os	bool		Terminal overstrikes.
pc	str		Pad character if not null.
pt	bool		Has hardware tabs; if they need to be set put sequence in **is** or **if**.
se	str		Ends stand out mode.
sf	str	(P)	Scrolls forwards.
sg	num		Number of blank chars left by **so** or **se**.
so	str		Begins stand out mode.
sr	str	(P)	Scroll reverse (backwards).
ta	str	(P)	Tab if not control-i or with padding.
tc	str		Name of terminal that has some of the same capabilities; **tc** must be the last capability.

Name	Type	Pad?	Description
te	str		Ends programs that do cursor motion.
ti	str		Initializes programs that do cursor motion.
uc	str		Underscores and moves past one character.
ue	str		Ends underscore mode.
ug	num		Number of blank spaces that surround underscore mode.
ul	bool		Terminal underlines automatically even though it can't overstrike.
up	str		Upline (cursor up).
us	str		Start underscore mode.
vb	str		Visible bell (must not move cursor).
ve	str		Ends open and visual modes.
vs	str		Initializes open and visual modes.
xb	bool		Beehive (f1=escape, f2=ctrl C).
xn	bool		Terminal ignores newline after wrap (Concept).
xr	bool		Return clears to end of line and goes to beginning of next line (Delta Data).
xs	bool		Writing on standout mode text produces standout mode text (HP 264?).
xt	bool		Destructive tabs, magic standout character (Teleray 1061).

Pointers on Preparing Descriptions

- You may want to copy the description of a similar terminal.
- Build up a description gradually, checking partial descriptions with *ex*.
- Be aware that an unusual terminal may expose bugs in *ex* or limitations in the *termcap* convention.

Basic Capabilities

The following capabilities are common to most terminals. The **co** capability gives the number of columns per line. The **li** capability gives the number of lines on a video terminal. The **am** capability indicates that writing off the right edge takes the cursor to the beginning of the next screen. The **cl** capability tells how the terminal clears its screen. The **bs** indicates that the terminal can backspace; but if the terminal doesn't use control-H, specify **bc** instead of **bs**. The **os** capability indicates that printing a character at an occupied position doesn't destroy the existing character.

A couple of notes on moving off the edge. Programs that use this convention never move the cursor off the top or the left edge of the screen. On the other hand, they assume that moving off the bottom edge scrolls the display up.

These capabilities suffice to describe hardcopy and very dumb terminals. For example, the Teletype Model 33 has this description.

```
        t3 ¦ 33 ¦ tty33:co#72:os
```

This is LSI ADM3 (without the cursor addressing option).

```
        cl ¦ adm3¦3¦lsi adm3:am:bs:cl=^Z:li#24:co#80
```

Cursor Addresses and Other Variables

If a string capability includes a variable value, use a **%** escape to indicate the value. By default, programs take these values to be zero origin (that is, the first possible value is 0) and that the **cm** capability specifies two values: row, then column. Use the **%r** or **%i** capability if either assumption is incorrect.

These are the valid % escapes.

%d	Print the values as a decimal number.
%2	Print the values as a two-digit decimal number.
%3	Print the values as a three-digit decimal number.
%.	Print the value in binary (but see below).
%+x	Add ASCII value of *x* to value, then print in binary.
%>xy	If the next value is greater than the ASCII value of *x*, add the ASCII value of *y* before using the value's % escape.
%r	Row is the first value in this **cm**.
%i	Values are 1-origin.
%%	Print a %.
%n	In this capability, exclusive OR the values with 01400 before using the values' % escapes (DM2500).
%B	Change the next value to binary coded decimal ((16*(*x*/10)) + (*x*%10) where *x* is the value) before interpreting it.

%D The next value is reverse-coded (*x*-2*(*x*%16) where *x* is the value; Delta Data).

A program should avoid using a **cm** sequence that includes a tab, newline, control-D, or return, because the terminal interface may misinterpret these characters. If possible, use the **cm** sequence to move to the row or column after the destination, then use local motion to get to the destination.

Here are some examples of **cm** definitions. To position the cursor of an HP2645 on row 3, column 12, you must send the terminal "\E&a12c03Y", followed by a 6 millisecond delay; the HP2645 description includes **:cm=6\E&%r%2c%2Y:**. To position the cursor of an ACT-IV, you send it a control-T, followed by the row and column in binary; the ACT-IV description includes **:cm=^T%.%.:**. The LSI ADM3a uses the set of printable ASCII characters to represent row and column values; its description includes **:cm\E=%+%+:**.

Local and General Cursor Motions

Most terminals have short strings that trigger commonly-used cursor motions. A non-destructive space (**nd**) moves the cursor one position right. An upline sequence (**up**) moves the cursor one position up. A home sequence (**ho**) moves the cursor to the upper left hand corner. A lower-left (**ll**) goes to the other lefthand corner. The **ll** capability may be a sequence that moves the cursor home, then up; but otherwise programs never do this.

Area Clears

Some terminals have short sequences that clear all or part of a display. Clear (**cl**) clears the screen and homes the cursor; if clearing the screen does not restore the terminal's normal modes, **cl** should include the strings that do. Clear to end of line (**ce**) clears from the current cursor position to the right. Clear to end of display (**cd**) clears from the current cursor position to the bottom of the display; programs always move the cursor to the beginning of the line before using **cd**.

Insert/Delete Line

Many terminals have strings that shift text starting at the current cursor position. Programs always move the cursor to the beginning of the line before using these strings. Add line (**al**) shifts the current line and all below it down a position leaving the cursor on the newly-blanked line. Delete line (**dl**) deletes the line the cursor is on without moving the cursor. If a terminal description has an **al** capability, you do not really need to specify **sb**.

If deleting a line might produce a non-blank line at the bottom of the screen, specify **db**. If scrolling backwards might produce a non-blank line at the top of the screen, specify **da**.

Insert/Delete Character

The *termcap* convention recognizes two kinds of terminal insert/delete string.

- The first convention is by far more common. Using insert or delete modes only affect characters on the current line. Inserting a single character shifts all characters, including all blanks, to the right; the character on the right edge of the screen is lost. No special capability is required to describe this kind of terminal.
- The second convention is less common and more complicated. The terminal distinguishes between blank spaces created by output tabs (011) or spaces (040) from all other blanks; other blanks are known as nulls. Inserting a character eliminates the first null to the right of the cursor; deleting a character doubles the first null. If there are no nulls on the current line inserting a character inserts the line's rightmost character at the beginning of the next line. Use the **in** capability to describe this kind of terminal.

Notably among the second type are the Concept 100 and Perkin Elmer Owl.

A simple experiment shows what type you have. Set the terminal to its "local" mode. Clear the screen, then type a short sequence of text. Move the cursor to the right several spaces *without using the space or tab characters*. Type a second short sequence of text. Move the cursor back to the beginning of the first text. Start the terminal's insert mode and begin tapping the space bar. If you have the first kind of terminal, both sequences of text will move at once, at whatever character is at the right edge of the screen will be lost. If you have the second kind of terminal, at first only the first sequence of text will move; when the first sequence hits the second sequence, it will push the second onto the next line.

A terminal can have either an insert mode or the ability to insert a single character. Specify insert mode with **im** and **ei**. To specify that the terminal can insert a single character, specify **ic** *and* specify empty strings for **im** and **ei**. If you must delay or output more control text after inserting a single character, specify **ip**.

If a terminal has both an insert mode and the ability to insert a single character, it is usually best not to specify **ic**.

Some programs operate more quickly if they are allowed to move the cursor around randomly while in insert mode. For example, *vi* has to delete a character when you insert a character before a tab. If your terminal permits this, specify move on insert (**mi**). Beware of terminals that foul up in subtle ways when you do this, notably Datamedia's.

Delete mode (**dm**), end delete mode (**ed**), and delete character (**dc**) work like **im**, **ei**, and **ic**.

Highlighting, Underlining, and Visible Bells

Specify the terminal's most distinctive display mode with **so** and **se**. Half intensity is usually not a good choice unless the terminal is normally in reverse video.

The convention provides for underline mode and for single character underlining. Specify underline mode with **us** and **ue**. Specify a way to underline and move past a character with **uc**; if your terminal can underline a single character but doesn't automatically move on, add a nondestructive space to the **uc** string.

Some terminals can't overstrike but still correctly underline text without special help from the host computer. If yours is one, specify **ul**.

If your terminal spaces before and after entering standout and underline mode, specify **ug**.

Programs leave standout and underline mode before moving the cursor or printing a newline.

If the terminal can flash the screen without moving the cursor, specify **vb** (visual bell).

If the terminal needs to change working modes before entering the open and visual modes of *ex* and *vi*, specify **vs** and **ve**, respectively. These can be used to change, for example, from a underline to a block cursor and back.

If the terminal needs to be in a special mode when running a program that addresses the cursor, specify **ti** and **te**. This may be important if a terminal has more than one page of memory. If the terminal has memory-relative cursor addressing but not screen relative cursor addressing, use **ti** to fix a screen-sized window into the terminal.

If a terminal can overstrike, programs assume that printable spaces don't destroy anything, unless you specify **eo**.

Keypad

Some terminals have keypads that transmit special codes. If the keypad can be turned on and off, specify **ks** and **ke**; if you don't, programs assume that the keypad is always on. Specify the codes sent by cursor motion keys with **kl**, **kr**, **ku**, **kd**, and **kh**. If there are function keys specify the codes they send with **f1**, **f2**, **f3**, **f4**, **f5**, **f6**, **f7**, **f8**, and **f9**. If these keys have labels other than the usual "f0" through "f9", specify the labels **l1**, **l2**, **l3**, **l4**, **l5**, **l6**, **l7**, **l8**, and **l9**. If there are other keys that transmit the same code that the terminal expects for a function, such as clear screen, mention the affected capabilities in the **ko** capability. For example, ":ko=cl,ll,sf,sb:" says that the terminal has clear, home down, scroll down, and scroll up keys that transmit the same thing as the **cl**, **ll**, **sf**, and **sb** capabilities.

Terminal Initialization

If a terminal must be initialized, on login for example, specify a short string with **is** or a file containing initialization strings with **if**. Other capabilities include **is**, an initialization string for the terminal, and **if**, the name of a file containing long initialization strings. If both are given, **is** is printed before **if**. If the terminal has tab stops, these strings should first clear all stops, then set new stops at the 9 column and every 8 columns thereafter.

Similar Terminals

If a new terminal strongly resembles an existing terminal, you can write a description of the new terminal that only mentions the old terminal and the capabilities that differ. The **tc** capability describes the old terminal; it must be the last capability in the description. If the old terminal has capabilities that the new one lacks, specify an @ *after the capability name*.

The different entries you create with **tc** need not represent terminals that are actually different. They can represent different uses for a single terminal, or user preferences as to which terminal features are desirable.

The following example defines a describes a variant of the *2621* that never turns on the keypad.

```
hn ¦ 2621nl:ks@:ke@:tc=2621:
```

FILES

/etc/termcap   standard data base

SEE ALSO

ex(1), tset(1), vi(1) in the *User's Reference Manual*.
curses(3X), otermcap(3X), terminfo(4).

BUGS

Ex allows only 256 characters for string capabilities, and the routines in *otermcap*(3X) do not check for overflow of this buffer.

The total length of a single description (excluding only escaped newlines) may not exceed 1024 characters. If you use **tc**, the combined description may not exceed 1024 characters.

The **vs**, and **ve** entries are specific to the *vi* program.

Not all programs support all entries. There are entries that are not supported by any program.

The **ma** capability is obsolete and serves no function in our database; Berkeley includes it for the benefit of systems that cannot run version 3 of *vi*.

[This page left blank.]

NAME

rfmaster - Remote File Sharing name server master file

DESCRIPTION

The **rfmaster** file is an ASCII file that identifies the hosts that are responsible for providing primary and secondary domain name service for Remote File Sharing domains. This file contains a series of records, each terminated by a newline; a record may be extended over more than one line by escaping the newline character with a backslash (\). The fields in each record are separated by one or more tabs or spaces. Each record has three fields:

name *type* *data*

The type field, which defines the meaning of the *name* and *data* fields, has three possible values:

p The **p** type defines the primary domain name server. For this type, *name* is the domain name and *data* is the full host name of the machine that is the primary name server. The full host name is specified as *domain.nodename*. There can be only one primary name server per domain.

s The **s** type defines a secondary name server for a domain. *Name* and *data* are the same as for the **p** type. The order of the **s** entries in the **rfmaster** file determines the order in which secondary name servers take over when the current domain name server fails.

a The **a** type defines a network address through which the previously mentioned name servers can be reached. *Name* is the full domain name for the machine and *data* is the network address of the "listener" service on that machine [see *nlsadmin*(1M)]. The network address can be in plain ASCII text or it can be preceded by a \x to be interpreted as hexadecimal notation. If the network address is preceded by a \$, it is interpreted as a command to be executed to obtain the address [see *getservaddr*(1M)]. (To determine the network addresses you need, see the documentation for the particular network you are using.)

There are at least two lines in the **rfmaster** file per domain name server: one **p** and one **a** line to define the primary name server and its network address. There should also be at least one secondary name server in each domain.

This file is created and maintained on the primary domain name server. When a machine other than the primary tries to start Remote File Sharing, this file is read to determine the address of the primary. If **rfmaster** is missing, the **-p** option of **rfstart** must be used to identify the primary. After that, a copy of the primary's **rfmaster** file is automatically placed on the machine.

Domains not served by the primary can also be listed in the **rfmaster** file. By adding primary, secondary, and address information for other domains on a network, machines served by the primary can share resources with machines in other domains.

A primary name server may be a primary for more than one domain. However, the secondaries must then also be the same for each domain served by the primary.

EXAMPLE

An example of an **rfmaster** file is shown below. (The network address examples are TCP network addresses.)

```
CT          p    CT.Convgt1
CT          s    CT.Convgt2
CT.comp2    a    \$ getservaddr Convgt1 nlsgen
CT.comp1    a    \$ getservaddr Convgt2 nlsgen
```

Note: If a line in the **rfmaster** file begins with a # character, the entire line is treated as a comment.

FILES

/usr/nserve/rfmaster

SEE ALSO

getservaddr(1M), rfstart(1M) in the *Administrator's Reference Manual*.

[This page left blank.]

NAME

system - system description file

MACHINE DEPENDENCY

DESCRIPTION

The system description describes tunable variables and hardware configuration to the System V system.

The file is formatted in sections. Each section begins with a section header (a ! followed by a single word). Each section varies in format, depending upon the format required by the program that uses the data provided by that section.

Note with respect to the **!TUNEABLES** section, that changes made to this section do not become effective until the *uconf*(1M) program is run.

In the example file below the **!TUNEABLES** section describes a cluster terminal configuration where only two cluster lines are being used and there are six ttys associated with each line: Cluster line 0 has tty256-261, and Cluster line 1 has tty262-267. (*uconf* must be run in order for this configuration to be effective.)

The **!VMESLOTS** section of the same example file describes the VME boards for the EEPROM. The slot field is the slot position in the VME bus. The type field is the board type; board types may be:

1	CMC Ethernet board
2	Interphase SMD disk controller board
4	Interphase 1/2-inch tape controller board
5	Multiprotocol Communications Controller board

The address field is the location of the board. The length field is the address space size of the board. The optional initialization function name is an initialization function that is called by the PROM at boot time.

The **!VMECODE** section consists of a list of files that describe the executable code to be loaded into the EEPROM. This section is required only if a bootable initialization function was specified.

The **!SCSIMAP** section consists of several lines, each line specifying a logical to physical mapping for a SCSI device [see *scsimap*(1M)].

EXAMPLE

```
!FILENAMES
PROM_IFILE=/etc/lddrv/EEPROM.ifile
EEPROM_FILE=/dev/vme/eeprom
!TUNEABLES
cl_deflines=2
cl_defdrops=6
!VMESLOTS
* The following section describes the VME boards
*
*slot   type    address          length  [Initialization
*                                        function name ]
*
0    2   C1000000         512      loadvs32
1    2   C1000200         512
*one CMC Ethernet controller)
2    1   CODE0000         131072
*
!VMECODE
/etc/lddrv/DISKVS32.o
!SCSIMAP   tape-d0  bus=0     target=0 lun=0   parity  reselect
tape-d1  bus=0      target=1  lun=0   parity  reselect
```

FILES

/etc/system

SEE ALSO

lddrv(1M), ldeeprom(1M), scsimap(1M), uconf(1M), vme(7).

[This page left blank.]

NAME

ttytype - list of terminal types by terminal number

DESCRIPTION

Ttytype is a text file that contains, for each terminal configured, the terminal type as described in *otermcap*(4). It is used by *tset*(1) when that program sets the **TERM** environment variable.

A line in *ttytype* consists of a terminal name (one of the abbreviations from the first field of the *termcap* entry), followed by a space, followed by the special file name of the terminal without the initial **/dev/**.

EXAMPLES

```
adm3 tty000
vt100   tty001
```

FILES

/etc/ttytype

SEE ALSO

tset(1) in the *User's Reference Manual*.
otermcap(4).

[This page left blank.]

NAME

TZ - time zone file

DESCRIPTION

The **/etc/TZ** file describes the time zone for the locality of the System V system. The file contains a single entry of the form:

*z*ST*n*[*z*DT]

where *z*ST is the standard three-letter abbreviation for the standard time zone; *n* is the difference in hours from Greenwich time; and *z*DT is the standard three-letter abbreviation for daylight saving time, if observed in the area.

The earth is divided into twenty-four (0 to 23) longitudinal standard time zones. Adjacent time zones are one hour (15 degrees) apart, beginning at Greenwich (0 degrees), with some variations in local legal time.

For the meridians of North America the principal time zones are:

AST4ADT	Atlantic Standard Time/Daylight Saving Time (60 degrees)
EST5EDT	Eastern Standard Time/Daylight Saving Time (75 degrees)
CST6CDT	Central Standard Time/Daylight Saving Time (90 degrees)
MST7MDT	Mountain Standard Time/Daylight Saving Time (105 degrees)
PST8PDT	Pacific Standard Time/Daylight Saving Time (120 degrees)
YST9YDT	Yukon Standard Time/Daylight Saving Time (135 degrees)
HST10HDT	Hawaiian Standard Time/Daylight Saving Time (150 degrees)
NST11NDT	Nome Standard Time/Daylight Saving Time (165 degrees)

FILES

/etc/TZ

SEE ALSO

ctime(3C), profile(4), timezone(4).
rc2(1M) in the *Administrator's Reference Manual*.

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* 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*	1/4	*lsubset*	⊆
oppE	∃	*3quarter*	3/4	*lsupset*	⊇
angstrom	Å	*degree*	°	*scrL*	ℒ
==<	≦	==>	≧		

FILES

/usr/pub/eqnchar

SEE ALSO

The DOCUMENTOR'S WORKBENCH REFERENCE MANUALS.

NAME

man - macros for formatting entries in this manual

SYNOPSIS

nroff -man files

troff -man [**-rs1**] 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. Note that this manual *is not* provided in online form as part of the System V software.

The default page size is 8.5"11", with a 6.5"10" text area; the **-rs1** option reduces these dimensions to 6"9" and 4.75"8.375", respectively; this option (which is *not* effective in *nroff*) also reduces the default type size from 10-point to 9-point, and the vertical line spacing from 12-point to 10-point. The **-rV2** option may be used to set certain parameters to values appropriate for certain Versatec printers: it sets the line length to 82 characters, the page length to 84 lines, and it inhibits underlining; this option should not be confused with the **-Tvp** option of the *man*(1) command, which is available at some UNIX System sites.

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, for example, **.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, for example, "local," *n* is new manual name. Invokes **.DT** (see below).
.SH *text*	Place subhead *text*, for example, **SYNOPSIS**, here.
.SS *text*	Place sub-subhead *text*, for example, **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 (for example, **.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

The DOCUMENTER'S WORKBENCH REFERENCE MANUALS.

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

me - macros for formatting papers

SYNOPSIS

nroff -me [options] file ...

troff -me [options] file ...

DESCRIPTION

This package of *nroff* and *troff* macro definitions provides a formatting facility for technical papers in various formats. When producing 2-column output on a terminal, filter the output through *col*(1).

The macro requests are defined below. Many *nroff* and *troff* requests are unsafe in conjunction with this package. The requests may be used after the first .pp, however:

.bp	begin new page
.br	break output line here
.sp *n*	insert *n* spacing lines
.ls *n*	(line spacing) *n* = **1** single, *n* = **2** double space
.na	no alignment of right margin
.cd *n*	center next *n* lines
.ul *n*	underline next *n* lines
.sz + *n*	add *n* to point size

Output of the *eqn*, *neqn*, *refer*, and *tbl*(1) preprocessors for equations and tables is acceptable as input.

REQUESTS

In the following list, "qinitialization" refers to the first .pp , .lp, .ip, .np, .sh, or .uh macro.

.(c	Begin centered block
.(d	Begin delayed text
.(f	Begin footnote
.(l	Begin list
.(q	Begin major quote
.(x *x*	Begin indexed item in index *x*
.(z	Begin floating keep
.)c	End centered block
.)d	End delayed text
.)f	End footnote
.)l	End list
.)q	End major quote
.)x	End index item

.)z	End floating keep
.+ + *m H*	Define paper section. *M* defines the part of the paper, and can be **C** (chapter), **A** (appendix), **P** (preliminary, for example, abstract, table of contents, and so forth), **B** (bibliography) **RC** (chapters renumbered from page one each chapter), or **RA** (appendix renumbered from page one).
.+c *T*	Begin chapter (or appendix, and so forth, as set by . + +). **T** is the chapter title.
.1c	One column format on a new page
.2c	Two column format.
.EN	Space after equation produced by *eqn* or *neqn*.
.EQ *x y*	Precede equation; break out and add space. Equation number is *y*. The optional argument *x* may be *I* to indent equation (default), *L* to left-adjust the equation, or *C* to center the equation.
.GE	End *gremlin* picture.
.GS	Begin *gremlin* picture.
.PE	End *pic* picture.
.PS	Begin *pic* picture.
.TE	End table.
.TH	End heading section of table.
.TS	Begin table; if *x* is **H** table has repeated heading.
.ac *A N*	Set up for ACM style output. *A* is the Author's name(s), *N* is the total number of pages. Must be given before the first initialization.
.b *x*	Print *x* in boldface; if no argument switch to boldface.
.ba +*n*	Augments the base indent by *n*. This indent is used to set the indent on regular text (like paragraphs).
.bc	Begin new column
.bi *x*	Print *x* in bold italics (nofill only)
.bu	Begin bulleted paragraph
.bx *x*	Print *x* in a box (nofill only)
.ef *´x´y´z´*	Set even footer to *x y z*
.eh *´x´y´z´*	Set even header to *x y z*

.fo *´x´y´z´*	Set footer to *x y z*
.hx	Suppress headers and footers on next page.
.he *´x´y´z´*	Set header to *x y z*
.hl	Draw a horizontal line
.i *x*	Italicize *x*; if *x* missing, italic text follows.
.ip *x y*	Start indented paragraph, with hanging tag *x*. Indentation is *y* ens (default 5).
.lp	Start left-blocked paragraph.
.lo	Read in a file of local macros of the form
.np	Start numbered paragraph
.of *´x´y´z´*	Set odd footer to *x y z*
.eh *´x´y´z´*	Set odd header to *x y z*
.pd	Print delayed text.
.pp	Begin paragraph. First line indented.
.r	Roman text follows.
.re	Reset tabs to default values.
.sc	Read in a file of special characters and diacritical marks. Must be given before initialization.
.sh *n x*	Section head follows, font automatically bold. *N* is level of section, *x* is title of section.
.sk	Leave the next page blank. Only one page is remembered ahead.
.sm *x*	Set *x* in a smaller pointsize.
.sz +*n*	Augment the point size by *n* points.
.th	Produce the paper in thesis format. Must be given before initialization.
.tp	Begin title page.
.u *x*	Underline argument (even in *troff*). (Nofill only).
.uh	Line .sh but unnumbered.
.xp *x*	Print index *x*.

FILES

/usr/lib/tmac/tmac.e
/usr/lib/me/*

SEE ALSO

The DOCUMENTER'S WORKBENCH REFERENCE MANUALS.

[This page left blank.]

NAME

mm - the MM macro package for formatting documents

SYNOPSIS

mm [options] [files]

nroff -mm [options] [files]

nroff -cm [options] [files]

mmt [options] [files]

troff -mm [options] [files]

troff -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

The DOCUMENTER'S WORKBENCH REFERENCE MANUALS

[This page left blank.]

NAME

mptx - the macro package for formatting a permuted index

SYNOPSIS

nroff -mptx [options] [files]

troff -mptx [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

The DOCUMENTER'S WORKBENCH REFERENCE MANUALS.

[This page left blank.]

NAME

ms - text formatting macros

SYNOPSIS

nroff -ms [options] file ...

troff -ms [options] file ...

DESCRIPTION

This package of *nroff* and *troff* macro definitions provides a formatting facility for various styles of articles, theses, and books. When producing 2-column output on a terminal or lineprinter, or when reverse line motions are needed, filter the output through *col*. All external **-ms** macros are defined below. many *nroff* and *troff* requests are unsafe in conjunction with this package. However, the first four requests below may be used with impunity after initialization, and the last two may be used even before initialization:

.bp	begin new page
.br	break output line
.sp *n*	insert *n* spacing lines
.ce *n*	center next *n* lines
.ls *n*	line spacing: *n* = 1 single, *n* = 2 double space
.na	no alignment of right margin

Font and point size changes with **\f** and **\s** are also allowed; for example, "\fIword\fR" will italicize *word*. Output of the *tbl*, *eqn*, and *refer* preprocessors for equations, tables, and references is acceptable as input.

REQUESTS

Macro Name	**Explanation**
.AB *x*	begin abstract; if *x* = **no** don't label abstract
.AE	end abstract
.AI	author's institution
.AM	better accent mark definitions
.AU	author's name
.B *x*	embolden *x*; if no *x*, switch to boldface
.B1	begin text to be enclosed in a box
.B2	end boxed text and print it
.BT	bottom title, printed at foot of page

.BX *x*	print word *x* in a box
.CM	cut mark between pages
.CT	chapter title: page number moved to CF (TM only)
.DA *x*	force date *x* at bottom of page; today if no *x*
.DE	end display (unfilled text) of any kind
.DS *x y*	begin display with keep; *x* = **I,L,C,B**; *y* = indent
.ID *y*	indented display with no keep; *y* = indent
.LD	left display with no keep
.CD	centered display with no keep
.BD	block display; center entire block
.EF *x*	even page footer *x* (3 page as for .tl)
.EH *x*	even page header *x* (3 part as for .tl)
.EN	end displayed equation produced by *eqn*
.EQ *x y*	break out equation; *x* = **L,I,C**; *y* = equation number
.FE	end footnote to be placed at bottom of page
.FP	numbered footnote paragraph; may be redefined
.FS *x*	start footnote; *x* is optional footnote label
.HD	optional page header below header margin
.I *x*	italicize *x*; if no *x*, switch to italics
.IP *x y*	indented paragraph, with hanging tag *x*; *y* = indent
.IX *x y*	index words *x y* and so on (up to 5 levels)
.KE	end keep of any kind
.KF	begin floating keep; text fills remainder of page
.KS	begin keep; unit kept together on a single page
.LG	larger; increase point size by 2
.LP	left (block) paragraph
.MC *x*	multiple columns; *x* = column width
.ND *x*	no date in page footer; *x* is date on cover
.NH *x y*	numbered header; *x* = level, *x* = **0** resets, *x* = **S** sets to *y*
.NL	set point size back to normal
.OF *x*	odd page footer *x* (3 part as for .tl)
.OH *x*	odd page header *x* (3 part as for .tl)
.P1	print header on 1st page
.PP	paragraph with first line indented
.PT	page title, printed at head of page.
.PX *x*	print index (table of contents); *x* = no suppresses title
.QP	quote paragraph (indented and shorter)
.R	return to Roman font
.RE	retreat: end level of relative indentation

.RP *x*	released paper format; *x* = no stops title on 1st page
.RS	right shift; start level of relative indentation
.SH	section header, in boldface
.SM	smaller; decrease point size by 2
.TA	set tabs to 8n 16n ... (*nroff*); 5n 10n ... (*troff*)
.TC *x*	print table of contents at end; x = no suppresses title
.TE	end of table processed by *tbl*
.TH	end multi-page header of table
.TL	title in boldface and two points larger
.TM	UC Berkeley thesis mode
.TS *x*	begin table; if *x* = **H** table has multi-page header
.UL *x*	underline *x*, even in troff
.UX *x*	UNIX; trademark message first time; *x* appended
.XA *x y*	another index entry; *x* = page or no for none; *y* = indent
.XE	end index entry (or series of .IX entries)
.XP	paragraph with first line exdented; others indented.
.XS *x y*	begin index entry; *x* = page or no for none; *y* = indent
.1C	one column format, on a new page
.2C	begin two column format
.]-	begin of refer reference
.[0	end of unclassifiable type of reference
.[*N*	*N* = **1**:journal-article, **2**:book, **3**:book-article, **4**:report

REGISTERS

Formatting distances can be controlled in **-ms** by means of built-in number registers. For example, this sets the line length to 6.5 inches:

```
.nr LL 6.5i
```

Here is a table of number registers and their default values:

Name	Register Controls	Takes Effect	Default
PS	point size	paragraph	10
VS	vertical spacing	paragraph	12
LL	line length	paragraph	6i
LT	title length	next page	same as LL
FL	footnote length	next .FS	5.5i
PD	paragraph distance	paragraph	1v(if n), .3v (if t)
DD	display distance	paragraph	1v(if n), .3v (if t)
PI	paragraph indent	paragraph	5n
QI	quote indent	next .QP	5n
FI	footnote indent	next .FS	2n
PO	page offset	next page	0 (if n), ¯1i (if t)
HM	header margin	next page 1i	
FM	footer margin	next page	1i
FF	footnote format	next .FS	0 (1, 2, 3 available)

When resetting these values, make sure to specify the appropriate units. Setting the line length to 7, for example, will result in output with one character per line. Setting FF to 1 suppresses footnote superscripting; setting it to 2 alwo suppresses indentation of the first line; and setting it to 3 produces an .IP-like footnote paragraph.

Here is a list of string registers available in **-ms**; they may be used anywhere in the text:

Name	String's Function
*Q	quote (" in nroff, “ in troff)
*U	unquote (" in nroff, ” in troff)
*-	dash (-- in nroff, – in troff)
*(MO	month (month of the year)
*(DY	day (current date)
**	automatically numbered footnote
*'	acute accent (before letter)
*`	grave accent (before letter)
*^	circumflex (before letter)
*,	cedilla (before letter)
*:	umlaut (before letter)
*~	tilde (before letter)

When using the extended accent mark definitions available with .AM, these strings should come after, rather than before, the letter to be accented.

FILES

/usr/lib/tmac/tmac.x
/usr/lib/ms/x.???

SEE ALSO

The DOCUMENTER'S WORKBENCH REFERENCE MANUALS.

BUGS

Floating keeps and regular keeps are diverted to the same space, so they cannot be mixed together with predictable results.

[This page left blank.]

NAME

mvt - 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, and so forth) 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).

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 ½ 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 (that is to say, 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

The DOCUMENTER'S WORKBENCH REFERENCE MANUALS.

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.