ULTRIX

digital

Reference Pages
Section 5: File Formats

Order Number: AA–LY18B–TE

ULTRIX

Reference Pages Section 5: File Formats

Order Number: AA-LY18B-TE
June 1990

Product Version: ULTRIX Version 4.0 or higher

This manual describes the format of system files and how the files are used on both RISC and VAX platforms.

digital equipment corporation
maynard, massachusetts

About Reference Pages

The *ULTRIX Reference Pages* describe commands, system calls, routines, file formats, and special files for RISC and VAX platforms.

Sections

The reference pages are divided into eight sections according to topic. Within each section, the reference pages are organized alphabetically by title, except Section 3, which is divided into subsections. Each section and most subsections have an introductory reference page called `intro` that describes the organization and anything unique to that section.

Some reference pages carry a one- to three-letter suffix after the section number, for example, `scan`(1mh). The suffix indicates that there is a "family" of reference pages for that utility or feature. The Section 3 subsections all use suffixes and other sections may also have suffixes.

Following are the sections that make up the *ULTRIX Reference Pages*.

Section 1: Commands

This section describes commands that are available to all ULTRIX users. Section 1 is split between two binders. The first binder contains reference pages for titles that fall between A and L. The second binder contains reference pages for titles that fall between M and Z.

Section 2: System Calls

This section defines system calls (entries into the ULTRIX kernel) that are used by all programmers. The introduction to Section 2, `intro`(2), lists error numbers with brief descriptions of their meanings. The introduction also defines many of the terms used in this section.

Section 3: Routines

This section describes the routines available in ULTRIX libraries. Routines are sometimes referred to as subroutines or functions.

Section 4: Special Files

This section describes special files, related device driver functions, databases, and network support.

Section 5: File Formats

This section describes the format of system files and how the files are used. The files described include assembler and link editor output, system accounting, and file system formats.

Section 6: Games

The reference pages in this section describe the games that are available in the unsupported software subset. The reference pages for games are in the document *Reference Pages for Unsupported Software*.

Section 7: Macro Packages and Conventions

This section contains miscellaneous information, including ASCII character codes, mail addressing formats, text formatting macros, and a description of the root file system.

Section 8: Maintenance

This section describes commands for system operation and maintenance.

Platform Labels

The *ULTRIX Reference Pages* contain entries for both RISC and VAX platforms. Pages that have no platform label beside the title apply to both platforms. Reference pages that apply only to RISC platforms have a "RISC" label beside the title and the VAX-only reference pages that apply only to VAX platforms are likewise labeled with "VAX." If each platform has the same command, system call, routine, file format, or special file, but functions differently on the different platforms, both reference pages are included, with the RISC page first.

Reference Page Format

Each reference page follows the same general format. Common to all reference pages is a title consisting of the name of a command or a descriptive title, followed by a section number; for example, `date`(1). This title is used throughout the documentation set.

The headings in each reference page provide specific information. The standard headings are:

Name	Provides the name of the entry and gives a short description.
Syntax	Describes the command syntax or the routine definition. Section 5 reference pages do not use the Syntax heading.
Description	Provides a detailed description of the entry's features, usage, and syntax variations.
Options	Describes the command-line options.
Restrictions	Describes limitations or restrictions on the use of a command or routine.
Examples	Provides examples of how a command or routine is used.

Return Values	Describes the values returned by a system call or routine. Used in Sections 2 and 3 only.
Diagnostics	Describes diagnostic and error messages that can appear.
Files	Lists related files that are either a part of the command or used during execution.
Environment	Describes the operation of the system call or routine when compiled in the POSIX and SYSTEM V environments. If the environment has no effect on the operation, this heading is not used. Used in Sections 2 and 3 only.
See Also	Lists related reference pages and documents in the ULTRIX documentation set.

Conventions

The following documentation conventions are used in the reference pages.

%	The default user prompt is your system name followed by a right angle bracket. In this manual, a percent sign (%) is used to represent this prompt.
#	A number sign is the default superuser prompt.
`user input`	This bold typeface is used in interactive examples to indicate typed user input.
`system output`	This typeface is used in text to indicate the exact name of a command, routine, partition, pathname, directory, or file. This typeface is also used in interactive examples to indicate system output and in code examples and other screen displays.
UPPERCASE lowercase	The ULTRIX system differentiates between lowercase and uppercase characters. Literal strings that appear in text, examples, syntax descriptions, and function definitions must be typed exactly as shown.
rlogin	This typeface is used for command names in the Syntax portion of the reference page to indicate that the command is entered exactly as shown. Options for commands are shown in bold wherever they appear.
filename	In examples, syntax descriptions, and routine definitions, italics are used to indicate variable values. In text, italics are used to give references to other documents.
[ ]	In syntax descriptions and routine definitions, brackets indicate items that are optional.
{ | }	In syntax descriptions and routine definitions, braces enclose lists from which one item must be chosen. Vertical bars are used to separate items.

. . .	In syntax descriptions and routine definitions, a horizontal ellipsis indicates that the preceding item can be repeated one or more times.
. . .	A vertical ellipsis indicates that a portion of an example that would normally be present is not shown.
`cat`(1)	Cross-references to the *ULTRIX Reference Pages* include the appropriate section number in parentheses. For example, a reference to `cat`(1) indicates that you can find the material on the `cat` command in Section 1 of the reference pages.

Online Reference Pages

The ULTRIX reference pages are available online if installed by your system administrator. The `man` command is used to display the reference pages as follows:

To display the `ls`(1) reference page:

```
% man ls
```

To display the `passwd`(1) reference page:

```
% man passwd
```

To display the `passwd`(5) reference page:

```
% man 5 passwd
```

To display the Name lines of all reference pages that contain the word "passwd":

```
% man -k passwd
```

To display the introductory reference page for the family of 3xti reference pages:

```
% man 3xti intro
```

Users on ULTRIX workstations can display the reference pages using the unsupported `xman` utility if installed. See the `xman`(1X) reference page for details.

Reference Pages for Unsupported Software

The reference pages for the optionally installed, unsupported ULTRIX software are in the document *Reference Pages for Unsupported Software*.

Name

intro – introduction to file formats

Description

This section describes the formats of various include files, program output files, and system files.

Name

a.out – assembler and link editor output

Syntax

#include <a.out.h>

Description

The `a.out` file contains the output from the assembler, `as`, and the link editor, `ld`. If errors and unresolved references do not exist, both programs make `a.out` executable. When submitted to the debugger, the `a.out` file provides symbolic information.

The RISC compilers and ULTRIX compilers use a file format that is similar to standard AT&T System V COFF (common object file format).

The RISC File Header definition is based on the AT&T System V header file *filehdr.h* with the following changes (also see `filehdr`(5)):

- The symbol table file pointer, *f_symptr,* and the number of symbol table entries, *f_nsyms,* specify the file pointer and the size of the Symbolic Header, respectively.
- All tables that specify symbolic information have their file pointers and number of entries in the Symbolic Header.

The Optional Header definition uses the same format as the System V header file, *aouthdr.h,* (the standard (pre-COFF) UNIX system a.out header) except the following fields have been added: *bss_start, gprmask, cprmask,* and *gp_value.*

The Section Header definition has the same format as the System V header file, *scnhdr.h,* except the line number fields (*s_lnnoptr* and *s_nlnno*) are used for gp tables (see `scnhdr`(5)).

The RISC relocation information definition is similar to that in Berkeley 4.3 UNIX, which has local relocation types (see `reloc`(5)).

The RISC file format is as follows:

- File Header
- Optional Header
- Section Headers
- Section Data — Includes text, read-only data, large data, 8- and 4-byte literal pools, small data, small bss (0 size), and large bss (0 size).
- Section Relocation Information — Includes information for text, read-only data, large data, 8- and 4-byte literal pools, and small data.
- Gp Tables — Missing if relocation information is not saved.
- Symbolic Header — Missing if fully stripped.
- Line Numbers — Created only if debugging is on and missing if stripped of nonglobals or fully stripped.
- Procedure Descriptor Table — Missing if fully stripped.

- Local Symbols — Missing if stripped of nonglobals or if fully stripped.
- Optimization Symbols — Created only if debugging is on and missing if stripped of nonglobals or fully stripped.
- Auxiliary Symbols — Created only if debugging is on and missing if stripped of nonglobals or fully stripped.
- Local Strings — Missing if stripped of nonglobals or if fully stripped.
- External Strings — Missing if fully stripped.
- Relative File Descriptors — Missing if stripped of nonglobals or if fully stripped.
- File Descriptors — Missing if stripped of nonglobals or if fully stripped.
- External Symbols — Missing if fully stripped.

See Also

as(1), ld(1), nm(1), dbx(1), strip(1), filehdr(5), scnhdr(5), reloc(5), syms(5), linenum(5).

Name

a.out – assembler and link editor output

Syntax

#include <a.out.h>

Description

The `a.out` file is the output file of the assembler `as`(1) and the link editor `ld`(1). Both programs make `a.out` executable if there were no errors and no unresolved external references. Layout information as given in the include file for the VAX is:

```
/*
* Header prepended to each a.out file.
*/
struct exec {
  unsigned short        a_magic;        /* magic number */
  unsigned short        a_mode;         /* mode parameter */
  unsigned    a_text;   /* size of text segment */
  unsigned    a_data;   /* size of initialized data */
  unsigned    a_bss;    /* size of uninitialized data */
  unsigned    a_syms;   /* size of symbol table */
  unsigned    a_entry;  /* entry point */
  unsigned    a_trsize; /* size of text relocation */
  unsigned    a_drsize; /* size of data relocation */
};

#define OMAGIC  0407    /* old impure format */
#define NMAGIC  0410    /* read-only text */
#define ZMAGIC  0413    /* demand load format */

/*
 *     Compatibility modes
 */
#define A_BSD   0               /* All pre V2.4 a.outs and BSD */
#define A_SYSV  1               /* System V compliant process */
#define A_POSIX 2               /* IEEE P1003.1 compliant process */

/*
* Macros which take exec structures as arguments and tell whether
* the file has a reasonable magic number or offsets
  to text | symbols | strings.
*/
#define       N_BADMAG(x) \
    ((x).a_magic)!=OMAGIC && ((x).a_magic)!=NMAGIC &&
    ((x).a_magic)!=ZMAGIC)

#define       N_TXTOFF(x) \
    ((x).a_magic==ZMAGIC ? 1024 : sizeof (struct exec))
#define N_SYMOFF(x) \
    (N_TXTOFF(x) + (x).a_text+(x).a_data + (x).a_trsize+(x).a_drsize)
#define       N_STROFF(x) \
    (N_SYMOFF(x) + (x).a_syms)
```

The file has five sections: a header, the program text and data, relocation information, a symbol table, and a string table (in that order). The last three sections may be omitted if the program was loaded with the **-s** option of `ld` or if the symbols and relocation have been removed by `strip`(1).

In the header, the sizes of each section are given in bytes. The size of the header is not included in any of the other sizes.

When an `a.out` file is executed, three logical segments are set up: the text segment, the data segment (with uninitialized data, which starts off as all 0, following initialized), and a stack. The text segment begins at 0 in the core image and the header is not loaded. If the magic number in the header is OMAGIC (0407), the number indicates that the text segment will not be write protected and shared, so the data segment is immediately contiguous with the text segment. This is the oldest kind of executable program and is rarely used.

If the magic number is NMAGIC (0410) or ZMAGIC (0413), the data segment begins at the first 0 mod 1024-byte boundary following the text segment, and the text segment is not writable by the program. If other processes are executing the same file, they will share the text segment. For ZMAGIC format, the text segment begins at a 0 mod 1024-byte boundary in the `a.out` file. The remaining bytes after the header in the first block are reserved and should be zero. In this case, the text and data sizes must both be multiples of 1024 bytes. The pages of the file will be brought into the running image as needed, and not preloaded as with the other formats. This is especially suitable for large programs and is the default format produced by `ld`(1).

The stack will occupy the highest possible locations in the core image, growing downwards from 0x7ffff000. The stack is automatically extended as required. The data segment is only extended as requested by `brk`(2).

After the header in the file, follow the text, data, text relocation, data relocation, symbol table, and string table in that order. The text begins at byte 1024 in the file for ZMAGIC format or just after the header for the other formats. The N_TXTOFF macro returns this absolute file position when given the name of an exec structure as argument. The data segment is contiguous with the text and immediately followed by the text relocation and then the data relocation information. The symbol table follows all this. Its position is computed by the N_SYMOFF macro. Finally, the string table immediately follows the symbol table at a position that can be easily accessed using N_STROFF. The first 4 bytes of the string table are not used for string storage; instead they contain the size of the string table which includes the 4 bytes. The minimum string table size is thus 4.

The layout of a symbol table entry and the principal flag values that distinguish symbol types are given in the include file as follows:

```
/*
* Format of a symbol table entry.
*/
struct nlist {
   union {
       char  *n_name;    /* for use when in-core */
       long   n_strx;    /* index into file string table */
    } n_un;
    unsigned char   n_type; /* type flag, i.e. N_TEXT; see below */
    char      n_other;
    short     n_desc;     /* see <stab.h> */
    unsigned n_value;     /* value of this symbol (or offset) */
};
#define n_hash  n_desc      /* used internally by ld */

/*
* Simple values for n_type.
```

```
*/
#define N_UNDF 0x0    /* undefined */
#define N_ABS  0x2    /* absolute */
#define N_TEXT 0x4    /* text */
#define N_DATA 0x6    /* data */
#define N_BSS  0x8    /* bss */
#define N_COMM 0x12   /* common (internal to ld) */
#define N_FN   0x1f   /* file name symbol */

#define N_EXT  01     /* external bit, or'ed in */
#define N_TYPE 0x1e   /* mask for all the type bits */

/*
* Other permanent symbol table entries have some N_STAB bits set.
* These are given in <stab.h>
*/
#define N_STAB 0xe0  /* if any of these bits set, don't discard */

/*
* Format for namelist values.
*/
#define N_FORMAT    "%08x"
```

In the `a.out` file, a symbol's n_un.n_strx field gives an index into the string table. An n_strx value of 0 indicates that no name is associated with a particular symbol table entry. The field n_un.n_name can be used to refer to the symbol name only if the program sets this up using n_strx and appropriate data from the string table.

If a symbol's type is undefined external, and the value field is nonzero, the symbol is interpreted by the loader `ld` as the name of a common region whose size is indicated by the value of the symbol.

The value of a byte in the text or data that is not a portion of a reference to an undefined external symbol is exactly the value that will appear in memory when the file is executed. If a byte in the text or data involves a reference to an undefined external symbol, as indicated by the relocation information, then the value stored in the file is an offset from the associated external symbol. 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 bytes in the file.

If relocation information is present, it amounts to 8 bytes per relocatable datum, as in the following structure:

```
/*
* Format of a relocation datum.
*/
struct relocation_info {
   int  r_address;    /* address which is relocated */
   unsigned   r_symbolnum:24,   /* local symbol ordinal */
        r_pcrel:1,    /* was relocated pc relative already */
        r_length:2,   /* 0=byte, 1=word, 2=long */
        r_extern:1,   /* does not include value of sym referenced */
         :4;          /* nothing, yet */
};
```

There is no relocation information if a_trsize+a_drsize==0. If r_extern is 0, then r_symbolnum is actually an n_type for the relocation (that is, N_TEXT meaning relative to segment text origin).

See Also

adb(1), as(1), dbx(1), ld(1), nm(1), strip(1), stab(5)

Name

acct – execution accounting file

Syntax

#include <sys/acct.h>

Description

The `acct`(2) system call makes entries in an accounting file for each process that terminates. The accounting file is a sequence of entries whose layout, as defined by the include file, is:

```
typedef u_short comp_t
struct  acct
{
        char    ac_comm[10];    /* Accounting command name */
        comp_t  ac_utime;       /* Accounting user time */
        comp_t  ac_stime;       /* Accounting system time */
        comp_t  ac_etime;       /* Accounting elapsed time */
        time_t  ac_btime;       /* Beginning time */
        short   ac_uid;         /* Accounting user ID */
        short   ac_gid;         /* Accounting group ID */
        short   ac_mem;         /* average memory usage */
        comp_t  ac_io;          /* number of disk IO blocks */
        dev_t   ac_tty;         /* control typewriter */
        char    ac_flag;        /* Accounting flag */
};

#define AFORK   0001    /* has executed fork, but no exec */
#define ASU     0002    /* used super-user privileges */
#define ACOMPAT 0004    /* used compatibility mode */
#define ACORE   0010    /* dumped core */
#define AXSIG   0020    /* killed by a signal */
#define AHZ     64      /* the accuracy of data is 1/AHZ */

#ifdef KERNEL
struct  acct    acctbuf;
struct  gnode   *acctp;
#endif
```

If the process does an `execve`(2), the first 10 characters of the file name appear in *ac_comm*. The accounting flag contains bits indicating whether `execve`(2) was ever accomplished and whether the process ever had superuser privileges.

See Also

acct(2), execve(2), sa(8)

Name

acct – execution accounting file

Syntax

#include <sys/acct.h>

Description

The acct(2) system call makes entries in an accounting file for each process that terminates. The accounting file is a sequence of entries whose layout, as defined by the include file, is:

```
struct  acct
{
        char    ac_comm[10];    /* Accounting command name */
        float   ac_utime;       /* Accounting user time */
        float   ac_stime;       /* Accounting system time */
        float   ac_etime;       /* Accounting elapsed time */
        time_t  ac_btime;       /* Beginning time */
        short   ac_uid;         /* Accounting user ID */
        short   ac_gid;         /* Accounting group ID */
        float   ac_mem;         /* average memory usage */
        float   ac_io;          /* number of disk IO blocks */
        dev_t   ac_tty;         /* control typewriter */
        char    ac_flag;        /* Accounting flag */
};

#define AFORK   0001    /* has executed fork, but no exec */
#define ASU     0002    /* used super-user privileges */
#define ACOMPAT 0004    /* used compatibility mode */
#define ACORE   0010    /* dumped core */
#define AXSIG   0020    /* killed by a signal */

#ifdef KERNEL
struct  acct    acctbuf;
struct  inode   *acctp;
#endif
```

If the process does an execve(2), the first 10 characters of the file name appear in *ac_comm*. The accounting flag contains bits indicating whether execve(2) was ever accomplished and whether the process ever had superuser privileges.

See Also

acct(2), execve(2), sa(8)

acucap (5)

Name

acucap – Automatic call unit capabilities file

Description

The `acucap` file lists the types of autodial modems and describes their attributes.

The `tip`(1c) program searches the `acucap` file when it encounters an `at` field in the `remote`(5) file description. If the `at` string matches a name entry in the `acucap` file, the `tip` and `uucp` generic dialing routines are used to place an outgoing call according to the attributes specified for the modem in the `acucap` file.

The `uucp`(1c) program uses the same procedure for deciding how to activate an autodialer modem, except that `uucp` searches for the brand name field of the `/usr/lib/uucp/L-devices` file in the `acucap` database.

Each line in the file describes how to dial a given type of modem. This description consists of strings, time delays, and flags that are used to control the action of any modem. Fields are separated by a colon (:). Entries that end in a backslash character (\) followed by a newline are continued on the next line.

The first entry is the name or names of the modem. If there is more than one name for a modem, the names are separated by vertical bars (|).

The fields of the description follow the name. A field name followed by an equal sign (=) indicates that a string value follows. A field name followed by a pound sign (#) indicates that a numeric value follows. A field name followed by the separating colon (:) represents a Boolean.

Options

The fields following the name of the modem define the capabilities of the modem. Capabilities are either strings (str), numbers (num), or Boolean flags (bool). A string capability is specified as *capability=value*; for example, "ss=^A^B". A numeric capability is specified by *capability#value*; for example, "sd#1". A Boolean capability is specified by simply listing the capability. Strings that are not specified cause nothing to be issued.

ab (str) Abort string. This string is sent to the modem if `tip`(1c) is aborted.

cd (num) Completion delay. This number gives the time to wait between completion string characters (in seconds, unless the `ls` Boolean is specified).

cr (bool) Setting this Boolean causes the program to wait in the generic dial routine until the modem senses a carrier.

co (str) A modem command string which instructs the modem to change from the default speed to the speed specified by the **xs** field.

cs (str) Completion string. The modem issues this string after receiving and responding to synchronization and dial strings.

da (num) Dial acknowledge. This number gives the time to wait before looking for a dial response (in seconds).

db (bool) Debug mode. Setting this Boolean causes the generic dialer to give more information as it dials.

dd (num) Dial delay. This number gives the time between dial characters (in seconds) unless the `ls` Boolean is specified.

di (str) Dial initialization. This string is used to start a dialing sequence (placed just before the number to dial.)

dr (str) Dial response. The modem sends this string if a dialing sequence that was just issued is successful.

ds (str) Disconnect string. This string is sent to the modem when `tip`(1c) is finally disconnected.

dt (str) Dial termination. This string is used to terminate a dialing sequence (placed just after the number to dial.)

fd (num) Full delay. This number is the time to wait for a carrier to be detected (in seconds). If the call is not completed in this time, an error is returned.

hu (bool) This Boolean causes the modem to hang up the phone if the line goes away.

is (num) This number specifies the speed the modem must be initialized at. The conversation speed will later be set as specified by the **xs** field.

ls (bool) Use an internal sleep routine rather than `sleep`(3) for delays. Thus, all delays are given in microseconds rather than seconds.

os (str) Online string. The modem sends this string after carrier has been detected.

rd (bool) Causes a 1-second delay after toggling dtr. This action will only be taken if the **re** flag is also set.

re (bool) This Boolean causes the modem to toggle dtr (data terminal ready) before beginning synchronization. It is used to reset the present condition of the modem.

rs (str) Replacement string. This string is a single character that will be substituted for an equal sign (=) or dash (-) in the number to be dialed. Used so that delay characters can be represented uniformly, but allowing the correct delay character to be passed to a given modem.

sd (num) Synchronization delay. This number gives the time between synchronization characters (in seconds unless the `ls` Boolean is specified.)

si (bool) This modem is attached to an interface that cannot return any characters until carrier is detected. Digital's DMF32 interface acts in this way.

sr (str) Synchronization response. What the modem sends in response to a synchronization string.

ss (str) Synchronization string. The first string the modem expects to receive; a check to see if the modem is operating.

xs (num) Specifies the speed the modem will operate at after initialization at the default speed per the **is** field.

Examples

The following example shows an entry for a Digital DF03 modem:

```
df03|DF03|dec df03:\
        :cr:hu:re:di=^A^B:dd#1:os=A:ds=^A:fd#40:
```

Files

`/etc/acucap`	Shared autodial modem data base

See Also

tip(1c), uucp(1c)

Name

aliases – aliases file for sendmail

Description

The aliases file is an ASCII file that describes user ID aliases that are used in `/usr/lib/sendmail`. It is formatted as a series of lines in the following form:

```
name: name_1, name2, name_3, . . .
```

The *name* is the name to alias, and the *name_n* are the aliases for that name. Each `alias` is separated from the next by a new line.

Continuation lines begin with white space. Comment lines begin with a number sign (#).

You can only assign aliases to local names. Loops are not allowed because a message should be sent to a person only once.

After an alias has been applied, local and valid recipients who have a `.forward` file in their home directory can have messages forwarded to the list of users defined in that file.

This is only the raw data file; the actual information pertaining to aliases is placed into binary format in the files `/etc/aliases.dir` and `/etc/aliases.pag` using the program `newaliases`(1). The `newaliases` command should be executed each time the aliases file changes. This command allows the new changes to take effect.

Restrictions

Because of restrictions in `dbm`(3x), a single alias cannot contain more than approximately 1000 bytes of information. You can specify longer aliases by chaining; that is, use a dummy name for the last name in the alias, which creates a continuation alias.

The `aliases` database may be distributed in a network by a naming service, such as Yellow Pages or BIND/Hesiod. See the *Guide to Yellow Pages* or the chapter on Hesiod in the *Guide to BIND* for setup information.

Files

`/etc/aliases`

See Also

newaliases(1), dbm(3x), sendmail(8)
"SENDMAIL Installation and Operation Guide", *ULTRIX Supplementary Documents,* Vol. III: System Manager
Guide to the BIND/Hesiod Service
Guide to the Yellow Pages Service

Name

ar – archive (library) file format

Syntax

#include <ar.h>

Description

The archive command, `ar`, combines several files into one. Archives are used mainly as libraries to be searched by the link-editor, `ld`.

A file produced by `ar` has a magic string at the start, followed by the constituent files, each preceded by a file header. The magic number and header layout as described in the include file are:

```
#define ARMAG   "!<arch>\n"
#define SARMAG 8

#define ARFMAG "`\n"

struct ar_hdr {
        char    ar_name[16];
        char    ar_date[12];
        char    ar_uid[6];
        char    ar_gid[6];
        char    ar_mode[8];
        char    ar_size[10];
        char    ar_fmag[2];
};
```

The name is a blank-padded string. The *ar_fmag* field contains ARFMAG to help verify the presence of a header. The other fields are left-adjusted, blank-padded numbers. They are decimal except for *ar_mode*, which is octal. The date is the modification date of the file at the time of its insertion into the archive.

Each file begins on an even (0 mod 2) boundary; a new-line is inserted between files if necessary. The size given reflects the actual size of the file exclusive of padding.

Provisions are not made for empty areas in an archive file.

The encoding of the header is portable across machines. If an archive contains printable files, the archive itself is printable.

Restrictions

A filename loses trailing blanks. Most software dealing with archives takes an included blank as a name terminator.

See Also

ar(1), ld(1), nm(1)

Name

ar – archive (library) file format

Syntax

#include <ar.h>

Description

The archive command, `ar`, combines several files into one. Archives are used mainly as libraries to be searched by the link-editor `ld`.

A file produced by `ar` has a magic string at the start, followed by the constituent files, each preceded by a file header. The magic number and header layout as described in the include file are:

```
#define       ARMAG  "!<arch>0
#define       SARMAG 8

#define       ARFMAG "`0

struct ar_hdr {
       char   ar_name[16];
       char   ar_date[12];
       char   ar_uid[6];
       char   ar_gid[6];
       char   ar_mode[8];
       char   ar_size[10];
       char   ar_fmag[2];
};
```

The name is a blank-padded string. The *ar_fmag* field contains ARFMAG to help verify the presence of a header. The other fields are left-adjusted, blank-padded numbers. They are decimal except for *ar_mode*, which is octal. The date is the modification date of the file at the time of its insertion into the archive.

Each file begins on a even (0 mod 2) boundary; a new-line is inserted between files if necessary. Nevertheless, the size given reflects the actual size of the file exclusive of padding.

There is no provision for empty areas in an archive file.

The encoding of the header is portable across machines. If an archive contains printable files, the archive itself is printable.

Restrictions

File names lose trailing blanks.

See Also

ar(1), ld(1), nm(1)

auth(5)

Name

auth – auth database

Description

The `auth` database is a repository of security-relevant information about each user of the system. This database contains the encrypted password associated with the user's account in addition to a list of assorted capabilities. The database is stored as an `ndbm`(3) database in the files `/etc/auth.pag` and `/etc/auth.dir`. Records are retrieved with the `getauthuid` library routine. Access to the database is restricted to the superuser and members of the group `authread`.

Auth records may be converted to an ASCII representation whose format is:

```
1000:4KvidFYwovnwp3j8lll78dC1:1920129:3600:2678400:03:0:1000:0:00:00
```

The first field is the UID of the entry that is used as the key into the database. Then follows:

Encrypted Password
: This is the user's encrypted password. Whether this password or the one from the `/etc/passwd` file is actually used is determined by the security level that the system is running at.

Password Modification Time
: This is the *time(2)* the password was last set.

Minimum Password Lifetime
: This is the minimum number of seconds which must elapse between setting passwords.

Maximum Password Lifetime
: This is the maximum period of time for which the password will be valid.

Account Mask
: These are capabilites pertaining to the account itself. They are:

  1 A_ENABLE: this account is enabled.
  2 A_CHANGE_PASSWORD: The user can change his or her password.
  4 A_ENTER_PASSWORD: The user is not required to use machine-generated passwords.

Login Failure Count
: This is the count of unsuccessful login attempts since the last successful login.

Audit ID
: Positive integer identifier used in generating audit records for the user.

Audit Control
: See the `audcntl`(2) reference page, SET_APROC_CNTL section for more information.

Audit Mask
: Determines which events will be audited for the user. See the `audcntl`(2) and `audit`(4) reference pages for more information.

Restrictions

Only the superuser and members of the group `authread` may read information from the auth database. Only the superuser may modify the auth database.

Files

```
/etc/auth.[pag,dir]
/etc/passwd
/etc/svc.conf
```

See Also

audcntl(2), getauthuid(3), getpwent(3), edauth(8)

CDA(5)

Name

CDA – Compound Document Architecture

Description

Digital's CDA architecture for compound documents is an open architecture that establishes a framework for the interchange of many types of data in a multivendor environment. Utilizing CDA converters, compound revisable format data can be handled much the same as ASCII text. With CDA converters, you can write applications that handle compound documents, regardless of the environment in which you or application users are working.

CDA includes the Digital Document Interchange Format (DDIF), the Data Object Transport Syntax (DOTS), and the Digital Table Interchange Format (DTIF). Each of these formats is encoded using the Digital Data Interchange Syntax (DDIS). Using these representations, CDA provides a method for manipulating files that contain a number of integrated components.

The tools associated with CDA include the CDA Toolkit (libddif.a), the CDA Converter (the main converter is `cdoc`(1)), and the CDA Viewers. The CDA Toolkit is a collection of routines that support the creation of CDA applications. The CDA Converter converts files of a specified input format to a specified output format. The CDA Viewers are used to display CDA-encoded files on a workstation display or character cell terminal.

All of the following products support CDA-encoded files. If you only intend to manipulate CDA files, and do not have an interest in the particulars of the file format, you can use any one of these products to manipulate a CDA-encoded file:

CDA Converters
CDA Viewers (dxvdoc, vdoc)
dxcardfiler
dxmail
dxpaint
PrintScreen

See Also

cdoc(1), vdoc(1), DDIF(5), DDIS(5), DOTS(5), DTIF(5)
Compound Document Architecture Manual

Name

core – format of memory image file

Syntax

#include <sys/param.h>

Description

When certain errors result in a terminated process, a `core` file is created that contains the memory image of a terminated process. A process can terminate for several reasons; however, the most common causes are memory violations, illegal instructions, bus errors, and user-generated quit signals. The `sigvec`(2) reference page contains a list of the causes.

The `core` is created in the working directory of the terminated process (normal access controls apply). The maximum size of a `core` cannot exceed the limit imposed by `setrlimit`(2).

The `core` file consists of the *u.* area, whose size (in pages) is defined by the UPAGES manifest in the *<sys/param.h>* file. The *u.* area starts with a *user* structure as given in *<sys/user.h>*. The remainder of the `core` file consists first of the data pages and then the stack pages of the process image. The amount of data space image in the `core` file is given (in pages) by the variable *u_dsize* in the *u.* area. The amount of stack image in the `core` file is given (in pages) by the variable *u_ssize* in the *u.* area. The size of a page is given by the constant NBPG (also from *<sys/param.h>*).

See Also

dbx(1), sigvec(2), setrlimit(2)

Name

core – format of memory image file

Syntax

#include <machine/param.h>

Description

The ULTRIX system writes out a memory image of a terminated process when any of various errors occur. See `sigvec`(2) for the list of reasons. The most common reasons are memory violations, illegal instructions, bus errors, and user-generated quit signals. The memory image is called `core` and is written in the process's working directory, provided it can be, and normal access controls apply.

The maximum size of a `core` file is limited by `setrlimit`(2). Files that would be larger than the limit are not created.

The core file consists of the *u.* area, whose size (in pages) is defined by the UPAGES manifest in the < machine/param.h > file. The *u.* area starts with a *user* structure as given in <sys/user.h>. The remainder of the core file consists first of the data pages and then the stack pages of the process image. The amount of data space image in the core file is given (in pages) by the variable *u_dsize* in the *u.* area. The amount of stack image in the core file is given (in pages) by the variable *u_ssize* in the *u.* area.

In general, the debugger `adb`(1) is sufficient to deal with core images.

See Also

adb(1), dbx(1), setrlimit(2), sigvec(2)

Name

cpio – format of cpio archive

Description

The header structure, when the `-c` option of `cpio`(1) is not used, is:

```
struct {
            short   h_magic,
                    h_dev;
            ushort  h_ino,
                    h_mode,
                    h_uid,
                    h_gid;
            short   h_nlink,
                    h_rdev
                    h_mtime[2],
                    h_namesize,
                    h_filesize[2]'
            char    h_name[h_namesize rounded to word];
} Hdr;
```

When the `-c` option is used, the header information is described by:

```
sscanf(Chdr,"%6o%6o%6o%6o%6o%6o%6o%11lo%6o%11lo%s",
        &Hdr.h_magic, &Hdr.h_dev, &Hdr.h_ino, &Hdr.h_mode,
        &Hdr.h_uid, &Hdr.h_gid, &Hdr.h_nlink, &Hdr.h_rdev,
        &Longtime, &Hdr.h_namesize,&Longfile,Hdr.h_name);
```

The *Longtime* and *Longfile* are equivalent to *Hdr.h_mtime* and *Hdr.h_filesize,* respectively. The contents of each file are recorded in an element of the array of varying length structures, *archive,* together with other items describing the file. Every instance of *h_magic* contains the constant 070707 (octal). The items *h_dev* through *h_mtime* have meanings explained in `stat`(2). The length of the null-terminated path name *h_name*, including the null byte, is given by *h_namesize*.

The last record of the *archive* always contains the name `TRAILER!!!` Special files, directories, and the trailer are recorded with *h_filesize* equal to zero.

See Also

cpio(1), find(1), stat(2)

crontab(5)

Name

crontab – clock daemon table file

Syntax

/usr/lib/crontab

Description

The `cron` command executes at specified dates and times according to the instructions in the `/usr/lib/crontab` file. The `crontab` file consists of lines with six fields each. The format for a line is as follows:

minute hour day month weekday command

The following list defines each field in the line:

minute **(0-59)**	The exact minute that the command sequence executes.
hour **(0-23)**	The hour of the day that the command sequence executes.
day **(1-31)**	The day of the month that the command sequence executes.
month **(1-12)**	The month of the year that the command sequence executes.
weekday **(1-7)**	The day of the week that the command sequence executes. Monday = 1, Tuesday = 2, and so forth.
command	The complete command sequence variable that is to be executed.

The first five integer fields may be specified as follows:

- A single number in the specified range
- Two numbers separated by a minus, meaning a range inclusive
- A list of numbers separated by commas, meaning any of the numbers
- An asterisk meaning all legal values

The sixth field is a string that is executed by the shell at the specified times. A percent sign (%) in this field is translated to a new-line character. Only the first line of the command field, up to a percent sign (%) or end of line, is executed by the shell. The other lines are made available to the command as standard input.

Examples

The following example is part of a `crontab` file:

```
# periodic things
0,15,30,45 * * * * (echo '^M' `date`; echo '') >/dev/console
0,15,30,45 * * * * /usr/lib/atrun

# daily stuff
5 4 * * * sh /usr/adm/newsyslog
15 4 * * * ( cd /usr/preserve; find . -mtime +7 -a -exec rm -f {} ; )
20 4 * * * find /usr/msgs -mtime +21 -a ! -perm 444 -a ! -name bounds
     -a -exec rm -f {} ;

# NOTE: The above line is wrapped.

# local cleanups
```

```
30 4 * * * find /usr/spool/mqueue -type f -mtime +5 -name df -exec rm {} ;
35 4 * * * find /usr/spool/mqueue -type f -mtime +5 -name tf -exec rm {} ;
40 4 * * * find /usr/spool/rwho -type f -mtime +21 -exec rm {} ;
#
```

Files

```
/etc/cron
/usr/lib/crontab
```

See Also

cron(8)
Guide to System Environment Setup

DDIF(5)

Name

DDIF – Digital Document Interchange Format (DDIF) files

Description

Digital Document Interchange Format (DDIF) is a DDIS/ASN.1 encoding for the interchange of revisable compound documents with document processing systems. DDIF is also a document output format, a storage format for user documents residing on a disk, and a compound document format.

The purpose of DDIF is to allow the creation of compound documents and also to serve as a standard intermediate format for the conversion of documents based on other formats. For example, a simple ASCII text file can be converted to DDIF, and the DDIF file can then be converted to PostScript. A DDIF document can also be converted to ASCII.

DDIF files are documents or portions of compound documents. A DDIF document is considered a simple document if it consists of one file. A DDIF document is considered a compound document if it consists of more than one file, the master of which must be a DDIF file.

A DDIF file can contain storage addresses (for example, filename) of other files, which must be DDIF, ASCII text, binary, or PostScript. References to DOTS files is not supported.

Because a DDIF file can reference another DDIF file and the referenced DDIF file can reference other DDIF files, a DDIF document can consist of a tree of files.

The following commands are used to manipulate DDIF files:

cdoc	Provides a set of converters to and from DDIF format.
ctod	Packs DDIF documents into DOTS syntax. The user can choose to archive a DDIF document in this manner. The `ctod` command also copies DDIF files from one location to another.
dtoc	Copies DDIF files from one location to another.
dxvdoc	Enables user to view DDIF documents. The `dxvdoc` command is used on workstations running ULTRIX UWS software. The `dxvdoc` command can also display imbedded graphics and image data that is encoded in the DDIF syntax.
vdoc	Enables user to view DDIF documents. The `vdoc` command is for use on character-cell terminals.

DDIF documents can be mailed to other users.

See Also

cdoc(1), ctod(1), dtoc(1), vdoc(1), dxvdoc(1X), DOTS(5), DDIS(5)

Name

DDIS – Digital Data Interchange Syntax / ISO ASN.1 (DDIS/ASN.1) files

Description

DDIS/ASN.1 files conform to Digital's Digital Data Interchange Syntax. DDIS conforms to syntaxes that can be defined within the specifications of International Standards Organization Abstract Syntax Notation One (ISO ASN.1), but is not itself an implementation of full ISO ASN.1 syntax.

DDIS/ASN.1 files conform to the DDIS/ASN.1 syntax. The DDIS/ASN.1 syntax is itself used to define other syntaxes. The following are among the syntaxes that are subsets of DDIS/ASN.1:

DDIF Digital Document Interchange Format

DTIF Digital Table Interchange Format

DOTS Data Object Transport Syntax

Files that conform to one of the DDIS/ASN.1 family of syntaxes are described as DDIS/ASN.1 files. More specifically, however, the files are typed according to a particular DDIS/ASN.1 syntax. For example, a file that conforms to the DDIF syntax is a DDIF file, and is identified by the `file`(1) command as a ddis/ddif file. The `file` command includes the string ddis/ as a part of its output if a file belongs in the DDIS family.

See Also

ctod(1), dtoc(1), DDIF(5), DOTS(5), DTIF(5), CDA(5)

dgateway (5)

Name

dgateway – name of the intermediate host (DECnet gateway)

Description

The `dgateway` file contains the ASCII name of the ULTRIX system serving as the intermediate host (gateway system) used by `dgate`(1c) to connect to the DECnet network. This gateway system must be connected to the local system through a local area (TCP/IP) network and to DECnet systems through the DECnet network.

The `dgate`(1c) command first looks in your home directory for a file named .dgateway. The .dgateway file should contain one (and only one) system, followed by a new line. In this case, you must have an account on the gateway system, as the `dgate`(1c) command will log in to this account to do its work there. Your gateway account must be set up so that a password is not required in order to gain access to the system. This is accomplished by means of a .rhosts file in your home directory on the remote system. See `rlogin`(1c) for more information about the .rhosts file.

If there is no .dgateway file in your home directory, the `dgate`(1c) command looks for the file `/etc/dgateway`, which has some optional additional fields separated by spaces:

```
gateway [ account ] [ path-to-dgated ]
```

In this case, the `dgate`(1c) command logs into the gateway system running `setuid` *guest*, while access is made through the *account* specified. The account specified must allow user *guest* to log in to the system without providing a password. This means that the .rhosts file in the home directory of the *account* listed must contain an entry of the form ''myhostname guest'', where "myhostname" is the name of the local system that desires access to the gateway node.

The syntax of the `~/.dgateway` file permits an optional username. The username on the gateway system must permit you to log in to that system from your system without using a password. For example:

home system: localhost
  `~/.dgateway` contains "remotehost username"

gateway system: remotehost
  `~username/.rhosts` contains "remotehost username"

This permits `dgate` to work even if the home system does not have an `/etc/dgateway` file, or if the gateway system does not have a guest account.

If no account is specified in the `/etc/dgateway` file, the default of the *guest* account is used. The last optional field specifies the pathname for the `dgated`(8) daemon. The default is `/etc/dgated`.

Files

```
/etc/dgateway
~/.dgateway
```

See Also

dgate(1c), dgated(8), rlogin(1c)

dir(5)

Name

dir – format of directories

Syntax

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

Description

A directory behaves exactly like an ordinary file, except that no user may write into a directory. The fact that a file is a directory is indicated by a bit in the flag word of its i-node entry. For further information, see `fs`(5). The structure of a directory entry is given in the include file.

A directory consists of some number of blocks of DIRBLKSIZ bytes, where DIRBLKSIZ is chosen such that it can be transferred to disk in a single atomic operation (for example, 512 bytes on most machines).

Each DIRBLKSIZ byte block contains some number of directory entry structures, which are of variable length. Each directory entry has a struct direct at the front of it, containing its inode number, the length of the entry, and the length of the name contained in the entry. These are followed by the name padded to a 4-byte boundary with null bytes. All names are guaranteed null terminated. The maximum length of a name in a directory is MAXNAMLEN.

The macro DIRSIZ(dp) gives the amount of space required to represent a directory entry. Free space in a directory is represented by entries which have dp->d_reclen > DIRSIZ(dp). All DIRBLKSIZ bytes in a directory block are claimed by the directory entries. This action usually results in the last entry in a directory having a large dp->d_reclen. When entries are deleted from a directory, the space is returned to the previous entry in the same directory block by increasing its dp->d_reclen. If the first entry of directory block is free, then its dp->d_ino is set to 0. Entries other than the first in a directory do not normally have dp->d_ino set to 0.

```
#ifdef KERNEL
#define DIRBLKSIZ DEV_BSIZE
#else
#define      DIRBLKSIZ 512
#endif

#define MAXNAMLEN 255
```

The DIRSIZ macro gives the minimum record length that will hold the directory entry. This requires the amount of space in struct direct without the d_name field, plus enough space for the name with a terminating null byte (dp->d_namlen+1), rounded up to a 4-byte boundary.

```
#undef DIRSIZ
#define DIRSIZ(dp) \
    ((sizeof (struct direct) - (MAXNAMLEN+1)) + \
    (((dp)->d_namlen+1 + 3) &~ 3))

struct direct {
       u_long d_ino;
       short  d_reclen;
       short  d_namlen;
```

```
        char   d_name[MAXNAMLEN + 1];
        /* typically shorter */
};

struct _dirdesc {
        int    dd_fd;
        long   dd_loc;
        long   dd_size;
        char   dd_buf[DIRBLKSIZ];
};
```

By convention, the first two entries in each directory are for dot (.) and dot dot (..). The first is an entry for the directory itself. The second is for the parent directory. The meaning of '..' is modified for the root directory of the master file system (''/''), where dot dot has the same meaning as dot.

See Also

fs(5)

disktab (5)

Name

disktab – disk description file

Syntax

#include <disktab.h>

Description

The `disktab` file is a simple data base that describes disk geometries and disk partition characteristics. The format is patterned after the `termcap`(5) terminal data base. Entries in `disktab` consist of a number of fields separated by colons (:). The first entry for each disk gives the names that are known for the disk, separated by vertical bars (|). The last name given should be a long name fully identifying the disk.

The following list indicates the normal values stored for each disk entry:

```
Name    Type    Description
ns      num     Number of sectors per track
nt      num     Number of tracks per cylinder
nc      num     Total number of cylinders on the disk
ba      num     Block size for partition 'a' (bytes)
bd      num     Block size for partition 'd' (bytes)
be      num     Block size for partition 'e' (bytes)
bf      num     Block size for partition 'f' (bytes)
bg      num     Block size for partition 'g' (bytes)
bh      num     Block size for partition 'h' (bytes)
fa      num     Fragment size for partition 'a' (bytes)
fd      num     Fragment size for partition 'd' (bytes)
fe      num     Fragment size for partition 'e' (bytes)
ff      num     Fragment size for partition 'f' (bytes)
fg      num     Fragment size for partition 'g' (bytes)
fh      num     Fragment size for partition 'h' (bytes)
pa      num     Size of partition 'a' in sectors
pb      num     Size of partition 'b' in sectors
pc      num     Size of partition 'c' in sectors
pd      num     Size of partition 'd' in sectors
pe      num     Size of partition 'e' in sectors
pf      num     Size of partition 'f' in sectors
pg      num     Size of partition 'g' in sectors
ph      num     Size of partition 'h' in sectors
se      num     Sector size in bytes
ty      str     Type of disk (e.g. removable, winchester)
```

The `disktab` entries can be automatically generated with the `diskpart` program.

Files

`/etc/disktab`

See Also

chpt(8), newfs(8)

Name

DOTS – Data Object Transport Syntax (DOTS) files

Description

Data Object Transport Syntax (DOTS) is DDIS/ASN.1 encoding for encapsulating the encoded interchange form of a number of related data objects. Data objects must be related by having embedded references to other objects in the same DOTS encapsulation. Typically, these embedded references depend on the storage address (for example, filename) of the referenced object. Therefore, when the referenced object is moved from one location to another, the storage address must be updated.

The purpose of DOTS is to allow composite data objects to be moved from one location to another as a single object and to allow the necessary storage reference to be updated as part of the process.

The primary use for DOTS is moving multifile compound documents in which one DDIF or DTIF file may have reference data stored in a physically separate file. Mail is a major vehicle for moving DOTS objects.

The commands `ctod` and `dtoc` are used to pack and unpack DDIF and DTIF files. The commands can also be used to copy a related set of DDIF or DTIF files, or both, from one location to another.

See Also

ctod(1), dtoc(1), DDIF(5), DTIF(5), DDIS(5), CDA(5)

DTIF(5)

Name

DTIF – Digital Table Interchange Format

Description

Digital Table Interchange Format (DTIF) is the standard format for the storage and interchange of documents that contain data tables, formulas, and spreadsheets. You can use DTIF to store and retrieve database information, interchange spreadsheets, and reference table data in compound documents.

DTIF defines the logical structure and physical layout of a data table, the values within the table (absolute data and/or expressions), and presentation attributes (formatting) to be used when displaying or printing the table. DTIF works with Digital Document Interchange Format (DDIF) so that you can store or reference DTIF tables in DDIF-encoded compound documents.

A DTIF document can contain a sequence of one or more tables and is uniquely identified by a product name, a version number, and other descriptive information such as the document's title and creation date. Each DTIF table is a 2-dimensional display of data values organized in columns and rows that has its own structure and table data stored in cells.

In DTIF documents, attributes specify the type and format of information pertaining to the data stored in a table. Column attributes describe information for all the cells in a particular column, whereas generic column attributes can be applied to any column in any table that references them. Format attributes define the printed and displayed presentation of data stored in the table. Format attributes can also be redefined at the window, column, or cell level.

See Also

CDA(5), DDIF(5), DTIF(5)
Compound Document Architecture Manual

Name

dumprestor, dumpdates – incremental dump format

Syntax

#include <sys/types.h>
#include <sys/inode.h>
#include <dumprestor.h>

Description

Tapes used by `dump` and `restore` contain:
A header record
Two groups of bit map records
A group of records describing directories
A group of records describing files

The format of the header record and of the first record of each description as given in the include file <dumprestor.h> is:

```
#define NTREC       10
#define MLEN        16
#define MSIZ        4096

#define TS_TAPE     1
#define TS_INODE    2
#define TS_BITS     3
#define TS_ADDR     4
#define TS_END      5
#define TS_CLRI     6
#define MAGIC       (int) 60011
#define CHECKSUM    (int) 84446

struct spcl {
       int          c_type;
       time_t       c_date;
       time_t       c_ddate;
       int          c_volume;
       daddr_t             c_tapea;
       ino_t        c_inumber;
       int          c_magic;
       int          c_checksum;
       struct       dinode       c_dinode;
       int          c_count;
       char         c_addr[BSIZE];
} spcl;

struct idates {
       char         id_name[16];
       char         id_incno;
       time_t       id_ddate;
};

#define      DUMPOUTFMT    "%-16s %c %s"       /* for printf */
                                    /* name, incno, ctime(date) */
#define      DUMPINFMT     "%16s %c %[^\n]\n"  /* inverse for scanf */
```

NTREC is the number of 1024-byte records in a physical tape block. MLEN is the number of bits in a bit map word. MSIZ is the number of bit map words.

The TS_ entries are used in the *c_type* field to indicate what sort of header this is. The types and their meanings are as follows:

TS_TAPE	Tape volume label.
TS_INODE	A file or directory follows. The *c_dinode* field is a copy of the disk inode and contains bits telling what sort of file this is.
TS_BITS	A bit map follows. This bit map has a one (1) bit for each inode that was dumped.
TS_ADDR	A subrecord of a file description. See *c_addr* described in the next list.
TS_END	End of tape record.
TS_CLRI	A bit map follows. This bit map contains a zero bit for all inodes that were empty on the file system when dumped.
MAGIC	All header records have this number in *c_magic*.
CHECKSUM	Header records checksum to this value.

The fields of the header structure are as follows:

c_type	The type of the header.
c_date	The date the dump was taken.
c_ddate	The date the file system was dumped from.
c_volume	The current volume number of the dump.
c_tapea	The current number of this (1024-byte) record.
c_inumber	The number of the inode being dumped if this is of type TS_INODE.
c_magic	This contains the value MAGIC above, truncated as needed.
c_checksum	This contains whatever value is needed to make the record sum to CHECKSUM.
c_dinode	This is a copy of the inode as it appears on the file system. For further information, see `fs`(5).
c_count	The count of characters in *c_addr*.
c_addr	An array of characters describing the blocks of the dumped file. A character is zero if the block associated with that character was not present on the file system; otherwise the character is nonzero. If the block was not present on the file system, no block was dumped; the block will be restored as a hole in the file. If there is not sufficient space in this record to describe all of the blocks in a file, TS_ADDR records will be scattered through the file, each one picking up where the last left off.

Each volume except the last ends with a tapemark (read as an end of file). The last volume ends with a TS_END record and then the tapemark.

The structure *idates* describes an entry in the file `/etc/dumpdates` where dump history is kept. The fields of the structure are:

id_name	The dumped filesystem is '/dev/*id_nam*'.
id_incno	The level number of the dump tape. For further information, see `dump`(8).
id_ddate	The date of the incremental dump in system format. For further information, see `types`(5).

Files

/etc/dumpdates

See Also

fs(5), types(5), dump(8), restore(8)

elcsd.conf(5)

Name

elcsd.conf – error logging configuration file

Description

The `elcsd.conf` file contains information used by the `elcsd` daemon to configure error logging for the system. The system manager maintains this file. The error logging daemon is dependent on the current order of the entries in the `elcsd.conf` file. Do not change the order.

The information in the `elcsd.conf` file shows any defaults and describes what you can enter. A newline is used to delimit each entry in the file, a null entry consists of a newline alone, and comments begin with #.

```
#
#    elcsd - errlog configuration file
#

{           # delimiter DON'T remove or comment out!
1           # status 1-local,2-logrem,4-remlog,5-remlog+priloglocal
            # errlog file size limit num. of blocks
/usr/adm/syserr # errlog dir. path
            # backup errlog dir. path
/           # single user errlog dir. path
/usr/adm/syserr # log remote hosts errlog dir. path
            # remote hostname to log to
}           # delimiter DON'T remove or comment out!
#  hosts to log :S - separate file or :R - remotes file (together)
remote1:S
remote2:S
#remote3:S          # disabled
remote4:S
   .
   .
   .
```

The status line of the `elcsd.conf` file describes where you can log error packets, also called error messages:

Logs error packets locally =
1, the default.

Logs error packets from a remote system or systems to the local machine =
2.

Logs local and remote error packets locally =
3.

Logs error packets from the local system to a remote system =
4.

Logs error packets from the local system remotely and logs high priority messages locally = **5.**

The errorlog file size defines the maximum size of an errorlog file. If disk space is limited, you can specify the maximum number of blocks (512 bytes each) you want the errorlog file to be. If you do not specify the maximum number of blocks, the system will notify you when the file system is 98% full.

The default errorlog directory path is `/usr/adm/syserr`. You can direct error packets to a different directory; if you do, you must change the default for `uerf` also. For further information, see `uerf`(8).

If the error-logging daemon cannot write to the primary errorlog directory path, it attempts to log to the backup errorlog directory path automatically.

The root directory is the default for the single-user errorlog directory path. When the system makes the transition to multiuser mode, errors logged in single-user mode are transferred to the default errorlog directory path `/usr/adm/syserr`. You can direct single-user error packets to another directory.

To log error packets from a remote system locally, set up an errorlog directory path on the local system. The default is `/usr/adm/syserr`.

Errorlog packets from remote systems can be logged to separate files or to one file. **S** sets up a separate errorlog file for each remote system that logs locally. **R** logs packets from the corresponding remote system to the file syserr.remotes. The default is **S**.

Restrictions

You must have superuser privileges to change the `elcsd.conf` file. However, anyone can view the file.

Files

`/usr/adm/elcsdlog`
elcsd daemon messages

See Also

elcsd(8), eli(8), uerf(8)
Guide to the Error Logger System

environ (5int)

Name

environ – natural language support (NLS) environment variables

Description

The international environment variables are defined for the ULTRIX system and are additional to those described in the ULTRIX reference pages, Sections 2 (system calls) and 3 (routines), and the `environ`(7) reference page. The international variables are made available to a process by `exec`.

This reference page is divided into two sections. The first section describes environment variables that can control the locale setting. The second section describes the variables that control where the `catopen` function searches for message catalogs and where the `setlocale` function searches for language databases.

Environment Variables That Control the Locale Setting

The LANG, LC_COLLATE, LC_TYPE, LC_NUMERIC, LC_TIME, and LC_MONETARY environment variables can control the locale setting. You define these variables using the same format as the *locale* argument to the `setlocale` function. The following shows the format you use:

```
language[_territory[.codeset]][@modifier]
```

In *language*, you specify the native language of the user. You can optionally specify the user's dialect and codeset using *_territory* and *codeset*. For example, the following definition of LANG specifies the French native language, as spoken in France (as opposed to in Switzerland), and the Digital Multinational Character Set:

```
LANG = FRE_FR.MCS
```

In *@modifier*, you specify a specific instance of localization data within a single category. For example, using *@modifier*, you can specify telephone directory ordering of data, as opposed to dictionary ordering of data. You cannot use *@modifier* to define the LANG variable.

The following list describes the environment variables that control the locale setting:

LANG Identifies the user's requirements for native language, local customs, and coded character set. At run time, you can bind the user's language requirements, as specified by the setting of `LANG`, to the execution of a program by calling `setlocale`, as follows:

```
setlocale (LC_ALL, "");
```

If `LANG` is not defined in the current environment, the locale defaults to the C locale. For more information on the C locale, see the *POSIX Conformance Document.*

System administrators can define `LANG` to provide a default setting for the system as a whole, or user's can define `LANG` individually using standard command interpreter facilities.

LC_COLLATE Contains the user's requirements for language, territory, and codeset for the character collation format. `LC_COLLATE` affects the behavior of regular expressions and the string collation

functions in `strcoll` and `strxfrm`. If `LC_COLLATE` is not defined in the current environment, `LANG` provides the necessary default.

LC_CTYPE Contains the user's requirements for language, territory, and codeset for the character classification and conversion format. `LC_CTYPE` affects the behavior of the character-handling functions in `conv` and `ctype`. If `LC_CTYPE` is not defined in the current environment, `LANG` provides the necessary default.

LC_MONETARY Contains the user's requirements for language, territory, and codeset for the monetary format. `LC_MONETARY` affects the currency string in `nl_langinfo`. If `LC_MONETARY` is not defined in the current environment, `LANG` provides the necessary default.

LC_NUMERIC Contains the user's requirements for language, territory, and codeset for the numeric data presentation format. `LC_NUMERIC` affects the radix and thousands separator character for the formatted I/O functions in `printf`, `scanf`, `nl_printf`, `nl_scanf`, and the string conversion functions in `ecvt` and `atol`. If `LC_NUMERIC` is not defined in the current environment, `LANG` provides the necessary default.

LC_TIME Contains the user's requirements for language, territory, and codeset for the time format. `LC_TIME` affects the behavior of the time functions in `strftime`. If `LC_TIME` is not defined in the current environment, `LANG` provides the necessary default.

Environment Variables That Specify Locations

The NLSPATH and INTLINFO environment variables control where the `catopen` and `setlocale` functions search for message catalogs and the language databases. You define these variables using a pathname or set of pathnames. The pathnames can contain variable elements, called substitution fields, that allow your program or the setting of other environment variables to affect the setting of NLSPATH and INTLINFO. The following shows the format you use to define these variables:

variable-name="[:][*/directory*][*/substitution field*]
[*/file-name*][*:alternate-pathname*][:...]"

You specify either NLSPATH or INTLINFO in place of *variable-name*.

A colon (:) that precedes other parts of any pathname in the definition specifies the current directory.

In *directory*, you can specify a specific directory in which the function searches. If you need the environment variable to be flexible, you can use a substitution field in place of or with directory names. A substitution field consists of a percent sign (%), followed by a code letter. The substitution fields you can use are as follows:

%N The value of the *name* argument you pass to `catopen`

%L The value of the LANG environment variable

%l The *language* element from LANG

%t The *territory* element from LANG

%c The *codeset* element from LANG

%% A literal percent sign

If a substitution field you specify is currently undefined, `catopen` or `setlocale` substitutes a null string. Neither function includes the underscore (_) or period (.) separator in `%t` or `%c` substitutions.

You can specify more than one pathname when you define these environment variables. You separate each pathname from the one that follows it using a colon (:). If you need to specify the current directory in a pathname other than the first pathname in the list, use a double colon (::). The functions interpret the first colon as a separator between pathnames and the second colon as specifying the current directory.

The following describes the ICONV, INTLINFO, and NLSPATH environment variables:

ICONV The ICONV environment variable stores the directory pathname for the conversion codesets used by the `iconv` command. If this variable is undefined, `iconv` searches the `/usr/lib/intln/conv` directory.

The following example shows how to define ICONV:

```
ICONV=/usr/lib/international/conversions
```

In this example, `ICONV` is defined as the directory pathname `/usr/users/international/conversions`.

INTLINFO

The INTLINFO environment variable stores the location of the language database. The `setlocale` function reads INTLINFO when it searches for the database.

The following example shows how to define INTLINFO:

```
INTLINFO = ":%L:/usr/lib/intln/%L:/usr/lib/intln/ENG_%t.%c"
```

In this example, the `setlocale` function searches for the language database named in the LANG environment variable. The function searches for the variable in the current directory. If the database is not in the current directory, `setlocale` searches in the `/usr/lib/intln` directory for that same database. Finally, if the database specified by LANG is unavailable, `setlocale` searches in `/usr/lib/intln` for the English language database that matches the current territory and codeset.

NLSPATH

The NLSPATH environment variable controls where the `catopen` function searches for a message catalog.

The following example shows defines NLSPATH:

```
NLSPATH=":%N.cat:/nlslib/%N.cat:nlslib/program.cat"
```

This definition causes `catopen` to search in the current directory for the message catalog named in the *name* argument you pass. If the function cannot find the message catalog in the current directory, it searches in the `/nlslib` directory. If the catalog is not in that directory, `catopen` opens the `/nlslib/program.cat` message catalog.

See Also

intro(3int), exec(2), setlocale(3), catopen(3int), lang(5int)
Guide to Developing International Software

ethers(5)

Name

ethers – database that maps Ethernet addresses to hostnames

Description

The `/etc/ethers` file is used in conjunction with the reverse address resolution protocol daemon, `rarpd`, to map Ethernet addresses to hostnames. It contains information about the known (48-bit) Ethernet addresses of hosts on the Internet.

For each host on an Ethernet, a single line should be present in the file with the following information:

Ethernet-address *official-host-name*

Items are separated by one or more spaces or tabs. A number sign (#) indicates the beginning of a comment that extends to the end of line.

The standard form for Ethernet addresses is:

x:x:x:x:x:x

The *x* is a hexadecimal number between 0 and ff, representing 1 byte. The address bytes are always in network order.

Hostnames can contain any printable character other than a space, tab, newline, or number sign (#).

Hostnames in the `/etc/ethers` file should correspond to the hostnames in the `/etc/hosts` file.

Examples

The following is a sample `ethers` file:

```
08:00:20:01:e5:1c       host1       # Comments go here
08:00:20:01:d0:4c       host2       # Comments go here
08:00:20:01:e0:1d       host3       # Comments go here
08:00:20:00:c2:4e       host4       # Comments go here
```

See Also

hosts(5), rarpd(8c)
Guide to Network Programming

Name

exports – defines NFS file systems to be exported

Syntax

/etc/exports

Description

The `/etc/exports` file describes the local file systems and directories that can be mounted by remote hosts through the use of the NFS protocol. The `exports` file can also be used to restrict access to a particular set of remote systems. The `mount` request daemon `mountd`(8nfs) accesses the `exports` file each time it receives a mount request from an NFS client.

Each entry in the `/etc/exports` file consists of a file system or directory name followed by an optional list of options or an optional list of identifiers or both. The identifiers define which remote hosts can mount that particular file system or directory. The identifiers listed beside the name of each file system or directory can be either host names or YP netgroups names. When the `mountd` daemon receives a mount request from a client, it searches for a match in the list of identifiers, first by checking the client host name with the host name identifiers and second by checking the client host name in a YP netgroups. When it finds a match, `mountd` makes that file system or directory available to the requesting client.

The exports file format is defined as follows:

```
pathname [-r=#] [-o] [identifier_1 identifier_2 ... identifier_n]
```

or

```
#anything
```

`pathname:` Name of a mounted local file system or a directory of a mounted local file system . The `pathname` must begin in column 1.

options:

–r=# Map client superuser access to uid #. If you want to allow client superusers access to the file system or directory with the same permissions as a local superuser, use `-r=0`. Use `-r=0` only if you trust the superuser on the client system. The default is `-r=2`, which maps a client superuser to nobody. This limits access to world readable files.

–o Export file system or directory read-only.

The options can be applied to both file system and directory entries in `/etc/exports`.

identifiers: Host names or netgroups, or both, separated by white space, that specify the access list for this export. Host names can optionally contain the local BIND domain name. For more information on BIND, see the *Guide to the BIND/Hesiod Service*.

NOTE

If no hosts or netgroups are specified, the `mount` daemon exports this file system or directory to anyone requesting it.

A number sign (#) anywhere in the line marks a comment that extends to the end of that line.

A whitespace character in the left-most position of a line indicates a continuation line.

Each file system that you want to allow clients to mount must be explicitly defined. Exporting only the root (/) will not allow clients to mount `/usr`. Exporting only `/usr` will not allow clients to mount `/usr/local`, if it is a file system.

Duplicate directory entries are not allowed. The first entry is valid and following duplicates are ignored.

Desired export options must be explicitly specified for each exported resource: file system or directory. If a file system and subdirectories within it are exported, the options associated with the file system are not ''inherited''. You do not need to export an entire file system to allow clients to mount subdirectories within it.

The access list associated with each exported resource identifies which clients can mount that resource with the specified options. For example, you can export an entire file system read-only, with a subdirectory within it exported read-write to a subset of clients. If a client that is not identified in the export access list of a directory attempts to mount it, then access is checked against the closest exported ancestor. If mount access is allowed at a higher level in the directory tree of the file system, the export options associated with the successful match will be in effect.

Examples

```
/usr alpha beta          # export /usr to hosts alpha and beta, client
                           superuser maps to uid -2 and read-write
                           access is permitted

/usr/staff/doe clients   # export directory to hosts in netgroup clients

/usr/man/man1 -o         # export directory read-only to everyone

/usr/local -r=0 beta     # export file system to beta, superuser
                           on beta maps to local superuser (uid=0)
```

Files

`/etc/exports`

See Also

hosts(5), mountd(8nfs), netgroup(5yp)
Guide to the BIND/Hesiod Service
Introduction to Networking and Distributed System

Name

filehdr – file header for RISC object files

Syntax

#include < filehdr.h>

Description

Every RISC 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 pointer to symbolic header */
   long            f_nsyms;     /* sizeof(symbolic header) */
   unsigned short  f_opthdr;    /* sizeof(optional header) */
   unsigned short  f_flags;     /* flags */
};
```

The byte offset into the file at which the symbolic header can be found is *f_symptr*. Its value can be used as the offset in `fseek`(3s) to position an I/O stream to the symbolic header. The ULTRIX system optional header is 56 bytes. The valid magic numbers are given below:

```
#define MIPSEBMAGIC 0x0160 /* objects for big-endian machines */
#define MIPSELMAGIC 0x0162 /* objects for little-endian machines */
#define MIPSEBUMAGIC 0x0180 /* ucode objects for big-endian machines */
#define MIPSELUMAGIC 0x0182 /* ucode objects for little-endian machines */
```

RISC object files can be loaded and examined on machines differing from the object's target byte sex. Therefore, for object file magic numbers, the byte-swapped values have define constants associated with them:

```
#define SMIPSEBMAGIC 0x6001
#define SMIPSELMAGIC 0x6201
```

The value in *f_timdat* is obtained from the `time`(2) system call. Flag bits used in RISC objects 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 */
```

See Also

time(2), fseek(3s), a.out(5)

Name

fs, inode – format of file system volume

Syntax

```
#include <sys/types.h>
#include <sys/fs.h>
#include <sys/inode.h>
```

Description

Every file system storage volume (disk, 9-track tape, for instance) has a common format for certain vital information. Every such volume is divided into a certain number of blocks. The block size is a parameter of the file system. Sectors 0 to 15 on a file system are used to contain primary and secondary bootstrapping programs.

The actual file system begins at sector 16 with the *super block*. The layout of the super block as defined by the include file *<sys/fs.h>* is:

```
#define FS_MAGIC     0x011954
struct   fs {
        struct   fs *fs_link;          /* linked list of file systems */
        struct   fs *fs_rlink;         /* used for incore super blocks */
        daddr_t fs_sblkno;             /* addr of super block in filesys */
        daddr_t fs_cblkno;             /* offset of cyl-block in filesys */
        daddr_t fs_iblkno;             /* offset of inode-blocks in filesys */
        daddr_t fs_dblkno;             /* offset of first data after cg */
        long     fs_cgoffset;          /* cylinder group offset in cylinder */
        long     fs_cgmask;            /* used to calc mod fs_ntrak */
        time_t   fs_time;              /* last time written */
        long     fs_size;              /* number of blocks in fs */
        long     fs_dsize;             /* number of data blocks in fs */
        long     fs_ncg;               /* number of cylinder groups */
        long     fs_bsize;             /* size of basic blocks in fs */
        long     fs_fsize;             /* size of frag blocks in fs */
        long     fs_frag;              /* number of frags in a block in fs */
/* these are configuration parameters */
        long     fs_minfree;           /* minimum percentage of free blocks */
        long     fs_rotdelay;          /* num of ms for optimal next block */
        long     fs_rps;               /* disk revolutions per second */
/* these fields can be computed from the others */
        long     fs_bmask;             /* ''blkoff'' calc of blk offsets */
        long     fs_fmask;             /* ''fragoff'' calc of frag offsets */
        long     fs_bshift;            /* ''lblkno'' calc of logical blkno */
        long     fs_fshift;            /* ''numfrags'' calc number of frags */
/* these are configuration parameters */
        long     fs_maxcontig;         /* max number of contiguous blks */
        long     fs_maxbpg;            /* max number of blks per cyl group */
/* these fields can be computed from the others */
        long     fs_fragshift;         /* block to frag shift */
        long     fs_fsbtodb;           /* fsbtodb and dbtofsb shift constant */
        long     fs_sbsize;            /* actual size of super block */
        long     fs_csmask;            /* csum block offset */
```

```
        long    fs_csshift;             /* csum block number */
        long    fs_nindir;              /* value of NINDIR */
        long    fs_inopb;               /* value of INOPB */
        long    fs_nspf;                /* value of NSPF */
        long    fs_sparecon[6];         /* reserved for future constants */
/* sizes determined by number of cylinder groups and their sizes */
        daddr_t fs_csaddr;              /* blk addr of cyl grp summary area */
        long    fs_cssize;              /* size of cyl grp summary area */
        long    fs_cgsize;              /* cylinder group size */
/* these fields should be derived from the hardware */
        long    fs_ntrak;               /* tracks per cylinder */
        long    fs_nsect;               /* sectors per track */
        long    fs_spc;                 /* sectors per cylinder */
/* this comes from the disk driver partitioning */
        long    fs_ncyl;                /* cylinders in file system */
/* these fields can be computed from the others */
        long    fs_cpg;                 /* cylinders per group */
        long    fs_ipg;                 /* inodes per group */
        long    fs_fpg;                 /* blocks per group * fs_frag */
/* this data must be recomputed after crashes */
        struct  csum fs_cstotal;        /* cylinder summary information */
/* these fields are cleared at mount time */
        char    fs_fmod;                /* super block modified flag */
        char    fs_clean;               /* file system is clean flag */
        char    fs_ronly;               /* mounted read-only flag */
        char    fs_flags;               /* currently unused flag */
        char    fs_fsmnt[MAXMNTLEN];
                                        /* name mounted on */
/* these fields retain the current block allocation info */
        long    fs_cgrotor;             /* last cg searched */
        struct  csum *fs_csp[MAXCSBUFS];
                                        /* list of fs_cs info buffers */
        long    fs_cpc;                 /* cyl per cycle in postbl */
        short   fs_postbl[MAXCPG][NRPOS];
                                        /* head of blocks for each rotation */
        long    fs_magic;               /* magic number */
        u_char  fs_rotbl[1];            /* list of blocks for each rotation */
/* actually longer */
};
```

Each disk drive contains some number of file systems. A file system consists of a number of cylinder groups. Each cylinder group has inodes and data.

A file system is described by its super block, which in turn describes the cylinder groups. The super block is critical data and is replicated in each cylinder group to protect against catastrophic loss. This is done at file system creation time and the critical super block data does not change, so the copies need not be referenced further unless disaster strikes.

Addresses stored in inodes are capable of addressing fragments of ''blocks''. File system blocks of at most size MAXBSIZE can be optionally broken into 2, 4, or 8 pieces, each of which is addressable; these pieces can be DEV_BSIZE or some multiple of a DEV_BSIZE unit.

Large files consist of exclusively large data blocks. To avoid undue wasted disk space, the last data block of a small file is allocated only as many fragments of a large block as are necessary. The file system format retains only a single pointer to such a fragment, which is a piece of a single large block that has been divided. The size of such a fragment is determinable from information in the inode, using the "blksize(fs, ip, lbn)" macro.

The file system records space availability at the fragment level; to determine block availability, aligned fragments are examined.

The root inode is the root of the file system. Inode 0 cannot be used for normal purposes and historically bad blocks were linked to inode 1; thus the root inode is 2. (Although inode 1 is no longer used for this purpose, numerous dump tapes make this assumption.) The *lost+found* directory is given the next available inode when it is initially created by *mkfs*.

fs_minfree gives the minimum acceptable percentage of file system blocks that may be free. If the freelist drops below this level, only the superuser can continue to allocate blocks. This can be set to 0 if no reserve of free blocks is deemed necessary; however, severe performance degradations will be observed if the file system is run at greater than 90% full. Thus, the default value of *fs_minfree* is 10%.

Empirically the best trade-off between block fragmentation and overall disk utilization at a loading of 90% comes with a fragmentation of 4. Thus, the default fragment size is a fourth of the block size.

Cylinder group related limits: Each cylinder keeps track of the availability of blocks at different rotational positions, so that sequential blocks can be laid out with minimum rotational latency. NRPOS is the number of rotational positions which are distinguished. With NRPOS 8, the resolution of the summary information is 2ms for a typical 3600 rpm drive.

fs_rotdelay gives the minimum number of milliseconds to initiate another disk transfer on the same cylinder. It is used in determining the rotationally optimal layout for disk blocks within a file; the default value for *fs_rotdelay* is 2ms.

Each file system has a statically allocated number of inodes. An inode is allocated for each NBPI bytes of disk space. The inode allocation strategy is extremely conservative.

MAXIPG bounds the number of inodes per cylinder group and is needed only to keep the structure simpler by having the only a single variable size element (the free bit map). MAXIPG must be a multiple of INOPB(fs).

MINBSIZE is the smallest allowable block size. With a MINBSIZE of 4096, it is possible to create files of size 2^32 with only 2 levels of indirection. MINBSIZE must be big enough to hold a cylinder group block, so changes to (struct cg) must keep its size within MINBSIZE. MAXCPG is limited only to the dimension of an array given in (struct cg); it can be made larger as long as that structure's size remains within the bounds dictated by MINBSIZE. Note that super blocks are never more than size SBSIZE.

The path name on which the file system is mounted is maintained in *fs_fsmnt*. MAXMNTLEN defines the amount of space allocated in the super block for this name. The limit on the amount of summary information per file system is defined by MAXCSBUFS. It is currently parameterized for a maximum of 2,000,000 cylinders.

Per cylinder group information is summarized in blocks allocated from the first cylinder group's data blocks. These blocks are read in from *fs_csaddr* (size *fs_cssize*) in addition to the super block. sizeof (struct csum) must be a power of 2 in order for the ''fs_cs'' macro to work.

Super block for a file system: MAXBPC bounds the size of the rotational layout tables and is limited by the fact that the super block is of size SBSIZE. The size of these tables is inversely proportional to the block size of the file system. The size of the tables is increased when sector sizes are not powers of 2, as this increases the number of cylinders included before the rotational pattern repeats (*fs_cpc*). The size of the rotational layout tables is derived from the number of bytes remaining in (struct fs).

MAXBPG bounds the number of blocks of data per cylinder group, and is limited by the fact that cylinder groups are at most one block. The size of the free block table is derived from the size of blocks and the number of remaining bytes in the cylinder group structure (struct cg).

Inode: The inode is the focus of all file activity in the UNIX file system. There is a unique inode allocated for each active file, each current directory, each mounted-on file, text file, and the root. An inode is 'named' by its device/i-number pair. For further information, see the include file *<sys/inode.h>*.

fstab (5)

Name

fstab – file containing static information about known file systems

Description

The /etc/fstab file contains descriptive information about the known file systems. By convention, /etc/fstab is created and maintained as a read-only file by the system administrator. Each file system is described by its own line within /etc/fstab. The order of these lines and the file systems they represent is important because fsck and mount sequentially process /etc/fstab in the performance of their tasks.

The format of each file system description in /etc/fstab is as follows:

```
spec:file:type:freq:passno:name:options
```

The meanings of these fields are:

spec The block special file name of the device on which the file system is located. It can also be a network name for nfs, such as /@erie or /@suez.

file The pathname of the directory on which the file system is mounted.

type How the file system is mounted. The ways in which a file system can be mounted are:
rw - mount the file system read-write
ro - mount the file system read only
rq - mount the file system read-write with quotas
sw - make the special file part of the swap space
xx - ignore the entry

freq The frequency (in days) with which the dump command dumps the rw, ro, and rq file systems.

passno The order in which the fsck command checks the rw, ro, and rq file systems at reboot time.

name The name of the file system type. File systems can have the following types: ufs -- ULTRIX file system and nfs -- SUN Network file system.

options The options field. This field contains an arbitrary string meaningful only when mounting file systems with the specified file system type name, such as NFS. The specific options are described in the mount reference pages.

Special actions occur for file systems of type sw and rq at system boot time. File systems of type sw are made part of the swap space by the swapon(8) command and disk quotas are automatically processed by the quotacheck(8) command and then enabled by the quotaon(8) command for rq file systems.

Examples

Here is a sample fstab file:

```
/dev/ra0a:/:rw:1:1:ufs::
/dev/ra1g:/usr:rw:1:2:ufs::
/@bigvax:/bigvax:rw:0:0:nfs::
```

```
/usr/uws2.0@bigvax:/usr/uws2.0:rw:0:0:nfs:soft,bg,nosuid:
/usr/dec@bigvax:/usr/dec:rw:0:0:nfs:bg,soft,nosuid:
/usr/pro/xyz@vax:/usr/pro/xyz:rw:0:0:nfs:bg,soft,intr,nosuid:
```

The last three entries in the `fstab` sample shown use NFS options as described in the `mount`(8nfs) reference page.

Restrictions

The passno field of the root file system should be specified as 1. Other file systems should have larger values. File systems on the same device should have distinct passno fields. File systems on different devices may have the identical passno fields to allow them to be simultaneously checked.

All field delimiters (:) must exist within each file system description; only the options field may not be present. However, only the fields spec and type are meaningful to sw file systems and only the type field is meaningful to xx file systems.

The file system description within `/etc/fstab` should be parsed only through use of the `getfsent` routines.

Files

`/etc/fstab`	File system information file

See Also

getfsent(3x), dump(8), fsck(8), mount(8), mount(8nfs), mount(8ufs) quotacheck(8), quotaon(8), swapon(8)

gettytab(5)

Name

gettytab – terminal configuration data base

Syntax

/etc/gettytab

Description

The `gettytab` file is a simplified version of the `termcap`(5) data base used to describe terminal lines. The initial terminal login process `getty`(8) accesses the `gettytab` file each time it starts, allowing simpler reconfiguration of terminal characteristics. Each entry in the data base is used to describe one class of terminal.

There is a default terminal class, *default*, that is used to set global defaults for all other classes. That is, the *default* entry is read, and then the entry for the class required is used to override particular settings.

Capabilities

Refer to `termcap`(5) for a description of the file layout. The *default* column below lists defaults obtained if there is no entry in the table obtained, nor one in the special *default* table.

Name	Type	Default	Description
ab	bool	false	Auto-baud speed selection mechanism
ap	bool	false	Terminal uses any parity
bd	num	0	Backspace delay
bk	str	0377	Alternate end of line character (input break)
cb	bool	false	Use crt backspace mode
cd	num	0	Carriage-return delay
ce	bool	false	Use crt erase algorithm
ck	bool	false	Use crt kill algorithm
cl	str	NULL	Screen clear sequence
co	bool	false	Console – add 0fter login prompt
ds	str	^Y	Delayed suspend character
ec	bool	false	Leave echo 2OFF
ep	bool	false	Terminal uses even parity
er	str	^?	Erase character
et	str	^D	End of text 2EOF character
ev	str	NULL	Initial environment
f0	num	unused	Tty mode flags to write messages
f1	num	unused	Tty mode flags to read login name
f2	num	unused	Tty mode flags to leave terminal as
fd	num	0	Form-feed (vertical motion) delay
fl	str	^O	Output flush character
hc	bool	false	Do not hangup line on last close
he	str	NULL	Hostname editing string
hn	str	hostname	Hostname
ht	bool	false	Terminal has real tabs

ig	bool	false	Ignore garbage characters in login name
im	str	NULL	Initial (banner) message
in	str	^C	Interrupt character
is	num	unused	Input speed
kl	str	^U	Kill character
lc	bool	false	Terminal has lower case
lm	str	login:	Login prompt
ln	str	^V	''literal next'' character
lo	str	/bin/login	Program to exec when name obtained
nd	num	0	Newline (line-feed) delay
nl	bool	false	Terminal has (or might have) a newline character
nx	str	default	Next table (for auto speed selection)
op	bool	false	Terminal uses odd parity
os	num	unused	Output speed
p8	bool	false	Use 8-bit characters
pc	str		Pad character
pd	bool	false	Disable parity on output
pe	bool	false	Use printer (hard copy) erase algorithm
pf	num	0	Delay between first prompt and following flush (seconds)
ps	bool	false	Line connected to a MICOM port selector
qu	str	^	Quit character
rp	str	^R	Line retype character
rw	bool	false	Do not use raw for input, use cbreak
sp	num	unused	Line speed (input and output)
su	str	^Z	Suspend character
tc	str	none	Table continuation
to	num	0	Timeout (seconds)
tt	str	NULL	Terminal type (for environment)
ub	bool	false	Do unbuffered output (of prompts and so forth)
uc	bool	false	Terminal is known upper-case only
we	str	^W	Word erase character
xc	bool	false	Do not echo control chars as ^X
xf	str	^S	XOFF (stop output) character
xn	str	^Q	XON (start output) character

If no line speed is specified, speed will not be altered from that which prevails when `getty` is entered. Specifying an input or output speed will override line speed for stated direction only.

Terminal modes to be used for the output of the message, for input of the login name, and to leave the terminal set as upon completion, are derived from the Boolean flags specified. If the derivation should prove inadequate, any (or all) of these three may be overriden with one of the `f0`, `f1`, or `f2` numeric specifications, which can be used to specify (usually in octal, with a leading 0) the exact values of the flags. Local (new tty) flags are set in the top 16 bits of this (32-bit) value.

Should `getty` receive a null character (presumed to indicate a line break), it will restart using the table indicated by the `nx` entry. If there is none, it will reuse its original table.

Delays are specified in milliseconds; the nearest possible delay available in the tty driver will be used. Should greater certainty be desired, delays with values 0, 1, 2, and 3 are interpreted as choosing that particular delay algorithm from the driver.

The `cl` screen clear string may be preceded by a (decimal) number of milliseconds of delay required (a la termcap). This delay is simulated by repeated use of the pad character `pc`.

The initial message, and login message, `im` and `lm` may include the character sequence `%h` to obtain the host name, `%t` to obtain the terminal name, and `%d` to obtain the date. (`%%` obtains a single percent (%) character.) The host name is normally obtained from the system, but may be set by the `hn` table entry. In either case, it can be edited with `he`. The `he` string is a sequence of characters; each character that is neither an at sign (@) nor a number sign (#) is copied into the final host name. An at sign (@) in the `he` string causes one character from the real host name to be copied to the final host name. A number sign (#) in the `he` string causes the next character of the real host name to be skipped. Surplus at signs (@) and number signs (#) are ignored.

When `getty` executes the login process, given in the `lo` string (usually `/bin/login`), it will have set the environment to include the terminal type, as indicated by the `tt` string, if it exists. The `ev` string can be used to enter additional data into the environment. It is a list of comma-separated strings, each of which should be of the form *name=value*.

If a nonzero timeout is specified with `to`, then `getty` will exit within the indicated number of seconds, either having received a login name and passed control to `login`, or having received an alarm signal, and exited. This may be useful to hang up dial-in lines.

The `p8` flag allows use of 8-bit characters.

The `pd` flag turns off parity on output. Output from `getty` is even parity unless the `op` flag, the `pd` flag, or the `p8` flag is specified. The `ap` flag is used to allow any parity on input. The `op(ep)` flag may be specified with the `ap` flag to allow any parity on input, but generate odd(even) parity on output. The parity on output is accomplished by using the eighth bit as the parity bit. `getty(8)` does not check parity of input characters in RAW mode or 8-bit mode.

Terminals that are set up to operate in 8-bit mode should use `gettytab` entries which include the `p8` flag. If a terminal that is set up in 8-bit mode fails to use an appropriate `gettytab` entry, the output from `getty` and `login` can appear as multinational characters. This is due to the fact that `getty` uses the eighth bit of characters to provide software generated parity. The software parity generation will transform certain ASCII characters into multinational characters. Earlier releases of the ULTRIX operating system did not display these multinational characters, due to the lack of full 8-bit support in the terminal subsystem.

Restrictions

Because some users insist on changing their default special characters, it is wise to define at least the erase, kill, and interrupt characters in the **default** table. In all cases, # or CTRL/H typed in a login name will be treated as an erase character, and @ will be treated as a kill character.

`login`(1) destroys the environment, so there is no point setting it in `gettytab`.

See Also

termcap(5), getty(8)

gfsi(5)

Name

gfsi – The Generic File System Interface

Description

The Generic File System Interface (GFSI) is the interface between the kernel and specific file system implementations such as `ufs`, the local ULTRIX file system and `nfs`, the Network File System. The Generic File System Interface has many performance improvements, along with a complete reorganization of the file system code. The GFS interface has been accomplished with modifications to both the mount table, `sys/mount.h` and to the inode, which under the GFS interface implementation is referred to as the gnode. The gnode is defined in the `sys/gnode.h` and `sys/gnode_common.h`.

The GFS interface allows superusers to `mount` and `umount` file systems on local and remote machines. Changes to the `/etc/fstab` file allow any type of mount to occur automatically at boot time in the files `/etc/rc` and `/etc/rc.boot`. Other than mounting and unmounting file systems, users should not see any difference in the local file system.

The GFS interface requires two system calls: `getmnt`(2) and `getdirentries`(2). The `getmnt` system call handles generic mounted file system data. The `getdirentries` system call handles generic directory entries from any file system.

See Also

getdirentries(2), getmnt(2), mount(2), fstab(5), nfs(5nfs), ufs(5), fsck(8), mount(8)

Name

group – group file

Description

The `group` file is an ASCII file that contains the following information for each group:

Group name
Encrypted password
Numerical group ID
Comma-separated list of all users allowed in the group

Each group name is separated from the next by a new line. The fields are separated by colons. If the password field is null, no password is demanded.

Because of the encrypted passwords, it can and does have general read permission and can be used, for example, to map numerical group IDs to names.

The `group` database can be distributed in a network by a naming service, such as Yellow Pages or BIND/Hesiod. See the *Guide to the Yellow Pages Service* or the *Guide to the BIND/Hesiod Service* for setup information.

Restrictions

The `passwd`(1) command will not change the passwords.

Files

`/etc/group`

See Also

passwd(1), setgroups(2), crypt(3), getgrent(3), initgroups(3x), passwd(5)
Guide to the BIND/Hesiod Service
Guide to the Yellow Pages Service

group(5yp)

Name

group – group file in a Yellow Pages environment

Description

For each group, the `group` file contains:

Group name
Encrypted password
Numerical group ID
Comma-separated list of all users allowed in the group

This is an ASCII file. The fields are separated by colons. Each group is separated from the next by a new-line. If the password field is null, no password is needed.

This file resides in the `/etc` directory. Because of the encrypted passwords, it can and does have general read permission and can be used, for example, to map numerical group ID's to names.

A group file can have a line beginning with a plus (+), which means to incorporate entries from the Yellow Pages. There are two styles of + entries: All by itself, + means to insert the entire contents of the Yellow Pages group file at that point; *+name* means to insert the entry (if any) for *name* from the Yellow Pages at that point. If a + entry has a password or group member field that is not null, the contents of that field will override what is contained in the Yellow Pages. The numerical group ID field cannot be overridden.

Examples

```
+myproject:::bill, steve
+:
```

If these entries appear at the end of a group file, then the group *myproject* will have members *bill* and *steve*, and the password and group ID of the Yellow Pages entry for the group *myproject*. All the groups listed in the Yellow Pages will be pulled in and placed after the entry for *myproject*.

Restrictions

The `passwd`(1) command will not change group passwords.

Files

`/etc/group`	ULTRIX file system group file
`/etc/yp/{domain}/group`	Yellow Pages group map

See Also

yppasswd(1yp), setgroups(2), crypt(3), initgroups(3x), passwd(5yp)

Name

hesiod.conf – Hesiod configuration file

Description

The Hesiod configuration file, `/etc/hesiod.conf`, contains information read by the Hesiod routines the first time they are invoked by a process. The Hesiod file consists of ASCII text.

The `/etc/hesiod.conf` file is required if your system is running BIND/Hesiod. This file must contain both the left-hand side and right-hand side of a Hesiod query.

The `/etc/hesiod.conf` file includes the two following entry formats:

`rhs` This line specifies the right-hand side of a Hesiod query. It consists of the BIND domain name preceded by a dot (.).

`lhs` This entry is not currently being used but exists for compatibility purposes.

This file is created by `bindsetup`(8).

Examples

The following is an example `hesiod.conf` file in the `dec.com` BIND domain.

```
rhs=.dec.com
lhs=
```

Files

`/etc/hesiod.conf`

See Also

hesiod(3), bindsetup(8)
Guide to the BIND/Hesiod Service

hosts (5)

Name

hosts – host name file

Description

The `hosts` file is an ASCII file that contains information about the known hosts on the DARPA Internet. For each host a single line should be present with the following information:

Internet address
Official host name
Aliases

Each `host` name is separated from the next by a new line. Items are separated by any number of blanks or tab characters. A number sign (#) indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines that search the file. This file is normally created from the official host data base maintained at the Network Information Control Center (NIC), though local changes may be required to bring it up to date regarding unofficial aliases or unknown hosts.

Host addresses are specified in the conventional dot (.) notation using the inet_addr routine from the Internet address manipulation library, `inet`(3n). Host names can contain any printable character other than a field delimiter, newline, or comment character.

The `hosts` database may be distributed in a network by a naming service, such as Yellow Pages or BIND/Hesiod. See the *Guide to the Yellow Pages Service* or the *Guide to the BIND/Hesiod Service* for setup information.

Files

`/etc/hosts`

See Also

gethostent(3n)
Guide to the BIND/Hesiod Service
Guide to the Yellow Pages Service

Name

hosts.equiv – list of trusted hosts

Description

The `hosts.equiv` file resides in the `/etc` directory and contains a list of trusted hosts. When an `rlogin`(1c) or `rsh`(1c) request from a host listed in the `hosts.equiv` file is made, and the initiator of the request has an entry in `/etc/passwd`, further validity checking is not required. Thus, `rlogin` does not prompt for a password, and `rsh` completes successfully. When a remote user is in the local `hosts.equiv` file, that user is defined as equivalenced to a local user with the same user ID.

The format of `hosts.equiv` is a list of names, as in:
host1
-host2
+@group1
-@group2

A line consisting of a host name means that anyone logging in from that host is trusted. A line consisting of a host name preceded by **-** means that anyone logging in from that host is not trusted. A line consisting of a single **+** means that all hosts are trusted.

NOTE

Placing a line consisting of a single **+** in your `hosts.equiv` file poses substantial security risks and is not recommended.

The **+@** and **-@** syntax are specific to Yellow Pages (YP). A line consisting of *+@group* means that all hosts in that network group (which is served by YP) are trusted. A line consisting of *–@group* means that hosts in that network group (which is served by YP) are not trusted. Programs scan the `hosts.equiv` file sequentially and stop when they encounter the appropriate entry (either positive for host name and +@ entries, or negative for –@ entries).

The `hosts.equiv` file has the same format as the `.rhosts` file. When a user executes `rlogin` or `rsh`, the `.rhosts` file from that user's home directory is concatenated onto the `hosts.equiv` file for permission checking. The host names listed in the `/etc/hosts.equiv` and `.rhosts` files may optionally contain the local BIND domain name. For more information on BIND, see the *Guide to the BIND/Hesiod Service*. If a user is excluded by a minus entry from `hosts.equiv` but included in `.rhosts`, that user is considered trusted. In the special case when the user is root, only the `.rhosts` file is checked.

It is possible to have two entries on a single line. Separate the entires with a space. If the remote host is equivalenced by the first entry, the user named by the second entry is allowed to specify any name to the `-l` option (provided that name is in the `/etc/passwd` file). For example:

```
suez john
```

This entry allows John to log in from suez. The normal use would be to put this entry in the `.rhosts` file in the home directory for *bill*. Then, John can log in as *bill* when coming from suez without having to supply a password. The second entry

can be a netgroup. For example:

```
+@group1 +@group2
```

This entry allows any user in *group2* coming from a host in *group1* to log in as anyone.

Files

/var/yp/*domain*/netgroup
/var/yp/*domain*/netgroup.byuser
/var/yp/*domain*/netgroup.byhost

See Also

rlogin(1c), rsh(1c), netgroup(5yp)
Guide to the BIND/Hesiod Service
Guide to the Yellow Pages Service

Name

inetd.conf – Internet daemon configuration data base

Description

The `inetd.conf` file contains information regarding the services that `inetd`(8c) will handle by opening sockets and listening for requests. For each service, a single line should be present with the following information:

Official service name (must be in `/etc/services`)

Socket type (stream or dgram)

Protocol name (must be in `/etc/protocols`)

Delay (wait or nowait)

Program (fully specified server program name)

Arguments (up to five arguments for server program)

Items are separated by any number of blanks or tab characters. A number sign (#) indicates the beginning of a comment. Characters up to the end of the line are not interpreted by routines that search the file.

Fields may contain any printable character other than a field delimiter, newline, or comment character.

A server marked as "wait" must be able to handle all requests that come to it during its lifetime. The `inetd`(8c) program will not invoke any new instances of the program until the current one terminates. If a server is marked as "nowait", a new invocation of the server will be started for every incoming request.

Files

`/etc/inetd.conf`

See Also

protocols(5), services(5), inetd(8c)

kitcap (5)

Name

kitcap – kit descriptor database for gentape and genra utilities.

Description

The `kitcap` file is a database for kit descriptors containing product codes, directories, files, and subsets that make up a product description to be used by `gentapes` or `genra` to create distribution media. All fields are separated by colons (:) with a backslash (\) at the end of a line indicating continuation. Lines starting with a number sign (#) are considered comments and are ignored. Comment fields with a kitcode description are delimited by an opening number sign (#) and a closing colon (:).

The following `kitcap` entry examples are for TK50 and MT9 media types:

```
Product-codeTK | Product Description:\
       directory1:directory2:directory3:\
       subset1:subset2:subset3:subset4:subset5

Product-codeMT | Product Description:\
       directory1:directory2:directory3:\
       subset1:subset2:subset3:subset4:subset5:\
       %%2:\
       subset6:subset7:subset8:subset9:subset10
```

The following parts make up the `kitcap` descriptor for magnetic tape media:

Product-code
: This is an arbitrary name made up of letters and/or numbers unique to the product that it describes. Typical codes include a product identifier and a version identifier, as indicated in the previous examples.

Media-code (TK or MT)
: The media-code is a 2 letter reference that describes the type of media the files will be written to. The media code must be either TK for TK50 or MT for 9-track magnetic tape devices. During run time, the `gentapes` utility probes the device to be written to and determines if it is a TK50- or MT9-type device. It then appends either TK or MT to the kitcode given on the command line and searches for the kitcode (product-code/media-code) in the kitcap file.

Product Description
: This field is a description of the software product that is being created by the `gentapes` utility and replaces the NAME field in the `.ctrl` file of all the subsets that make up a product. This is an optional field for magnetic tape media.

Directories
: The magnetic tape media production utility has the ability of producing multi-product tapes. That is, it can take subsets from different products that are based in different directories and merge them together to form a third product, which is a combination of the original products. Directory entries provide the full path locations of where the subsets that are to be put on media will be stored. There must be at least one directory entry for each kitcap descriptor.

Subsets

This field provides a list of subsets or files that are to be either written to the magnetic tape media or verified from the magnetic tape media. Each subset listed must be stored in one of the directories listed in that particular kitcap descriptor. If a file or subset is stored in a subdirectory of one of the directories listed in the kitcap descriptor, it is possible to include that sub-path with the subset/filename entry instead of listing the entire path/subpath as another directory listing. For example, a directory listed in the kitcap descriptor under the rules given in the Directories section is listed as:

```
/KITS/MYPRODUCT/001
```

A particular subset or file that a user would like to include on the media is stored in:

```
/KITS/MYPRODUCT/001/subdirectory/subset
```

Since the subdirectory/subset specification is part of the `/KITS/MYPRODUCT/001` directory tree, it is not necessary to include the full path `/KITS/MYPRODUCT/001/` subdirectory in the directory listing. An alternative is to include the subdirectory path with the subset name in the subset list. For example:

```
MY-PROD-001 | This is a good product:\
       /KITS/MYPRODUCT/001:\
       subset1:subset2:subdirectory/subset3:subset4
```

Volume identifier (MT9 media only)

The volume identifier is optional. Multi-tape support is available for products that have subsets or files that take up more room than is available by a single 9-track magnetic tape. If the subset list results in an end-of-tape condition, the subset list can be split into any number of multi-volume sets by placing `%%n` (where n is the volume number of the next tape) anywhere appropriate in the subset list. The subsets listed between the volume identifiers must fit on a single piece of media. By default, the subset list located directly after the directory list is always considered the first volume. Therefore, a volume identifier for the first volume in a multi-volume kit descriptor is not necessary.

The following example shows a `kitcap` entry for disks:

```
Product-codeRA:partition:\
       dd=/:Product_Description:\
       directory1:directory2:directory3:\
       subset1:subset2:subset3:subset4:subset5:\
       dd=SUB/DIR:Product_Description:\
       directory1:directory2:directory3:\
       subset1:subset2:subset3:subset4:subset5
```

The following parts make up the kitcap descriptor for disk media:

Product-code

Same as for magnetic tape.

Media-code

The media code for disks is RA and is appended to the product-code provided by the user at run time, by the `genra` utility.

Disk Partition
: This field is the partition where you want the software written to on the disk.

dd=
: This field tells the genra utility what directory you want the subsets written to on the disk media that is being created. The contraction dd can be thought of as the ''destination directory'' for the subsets. This field is required and allows a hierarchial structure for those who want to put multiple products on the same disk, or want to separate parts of one product into different areas on the disk.

Typically, a disk is mounted by the genra utility onto a temporary mount point under /usr/tmp This location becomes the disks root directory. If a user wants to have only one directory for an entire product, a valid entry would be dd=/. This entry tells the genra utility to write all the following subsets under the mount point.

In the disk kitcap descriptor example given previously, the first five subsets are being written to the mount point, or root directory, for the disk media being made. Then a new directory on the disk media is made, /mnt_point/SUB/DIR, and the next five subsets are written into that directory on the disk media.

It is important to note that the top-level directory of the media disk is always considered the mount point used by the genra script and is referenced by dd=/. Any subdirectories listed as destination directories are created starting from the mount point and must be referenced in full. For instance, in the previous example, if the user wanted to put some other subsets in a subdirectory of DIR, the entry would be dd=SUB/DIR/SUBSUBDIR. Note that each new destination directory requires a product description.

Product Description
: This field is similar to the one defined under the magnetic tape description. However, in the case of disk media there are 2 important differences. The product description is a required field, and all words in the description must be connected with underscores (_). The genra script removes the underscores at run time. For example, suppose the desired description was as follows:

```
This is a good product
```

The Product Description entry when making disk media would become:

```
This_is_a_good_product
```

Directories
: Same as for magnetic tape.

Subsets
: Same as for magnetic tape.

Examples

TK50 and MT9 (single-volume tape)kitcap description

```
MYPRODUCT400 | MYPRODUCT software version 4:\
       :# directory listing :\
       /directory1:/directory2:/directory3:\
       :# subset listing :\
       subset1:subset2:subset3:subset4:subset5
```

MT9 kitcap description (multi-volume tape)

```
MYPRODUCT400 | MYPRODUCT software version 4:\
        /directory1:/directory2:/directory3:\
        subset1:subset2:subset3:subset4:subset5:\
        :# Volume 2 :\
        %%2:\
        subset6:subset7:subset8:subset9:subset10
```

RA60 kitcap description (single product)

```
MYPRODUCT400:c:\
        dd=/:MYPRODUCT_software_version_4:\
        /directory1:/directory2:/directory3:\
        subset1:subset2:subset3:subset4:subset5
```

RA60 kitcap description (multiple product)

```
MYPRODUCT400:c:\
        dd=MYPRODUCT/BASE:\
        MYPRODUCT_software_version_4_base_subsets:\
        /directory1:/directory2:/directory3:\
        subset1:subset2:subset3:subset4:subset5:\
        dd=MYPRODUCT/NONBASE:\
        MYPRODUCT_software_version_4_nonbase_subsets:\
        /directory1:/directory2:/directory3:\
        subset1:subset2:subset3:subset4:subset5
```

See Also

genra(8), gentapes(8)

krb.conf(5krb)

Name

krb.conf – Kerberos configuration file

Syntax

/etc/krb.conf

Description

The `krb.conf` file contains configuration information describing the Kerberos realm and the Kerberos servers for each realm.

The first line of the `/etc/krb.conf` file contains the name of the realm for the local host. The following lines of the file indicate additional realm/host entries. These lines can contain two parts. The first part is the realm name; the second part is the host running a Kerberos server for that realm.

Examples

The following example shows a Kerberos configuration file:

```
dec.com
dec.com    mercury.dec.com
dec.com    venus.dec.com
dec.com    earth.dec.com
```

Files

`/etc/krb.conf`

See Also

krb_get_lrealm(3krb)

Name

krb_dbase – ASCII version of the Kerberos database

Description

All of the Kerberos tools, including the `kerberos` daemon, access a version of the Kerberos database that is stored in an `ndbm`-formatted file. See the `ndbm(3)` reference page for more information. Files in `ndbm` format are not user readable. To examine the Kerberos database, it is necessary to convert the `ndbm` database into an ASCII-formatted file with `kdb_util(8krb)`. A file in `krb_dbase` format is an ASCII-formatted version of the Kerberos database.

Each line in a `krb_dbase`-formatted file lists the attributes associated with a single Kerberos principal. The following list describes the fields as they appear from left to right in a `krb_dbase` file. A blank entry in the database is indicated by an asterisk (*).

Kerberos primary name: The primary name is the first part of the principal name that the line describes. It is usually equivalent to the name of the application or user that is associated with the principal.

Kerberos instance name: The instance name is the second section of the principal name that the line describes. It is usually equivalent to the name of the machine on which an application runs. If the primary name references a user, the instance name is blank.

Maximum ticket lifetime: The third entry is the maximum lifetime of a ticket produced for the principal by the ticket-granting service. The number stored in the `krb_dbase` file indicates the number of 5-minute intervals for which the ticket is valid. For example, if the maximum ticket lifetime of a principal is 10, any ticket that the principal acquires from the ticket-granting service will expire in a maximum of 50 minutes. The maximum ticket lifetime corresponds to a value of 255 (21 hours and 15 minutes).

Kerberos database key version: The master key of the Kerberos database is used to encrypt sections of the Kerberos database. This master key can be changed. The fourth entry is the version number associated with the master key of the Kerberos database.

Principal key version: The key associated with the principal can also change. The fifth field records the version number of the key associated with the principal.

Attributes: The attributes field is not currently used by the ULTRIX implementation of Kerberos. It should always be zero.

Key of the principal: The key of the principal is stored in the seventh and eighth fields. It is encrypted with the master database key.

Expiration time: The date on which the principal's entry in the Kerberos database will expire is stored in the ninth field. The first four digits of the date indicate the year in which the entry will expire. The next two digits indicate the month, the seventh and eighth digits indicate the day, and the last four digits indicate the hour and minute at which the entry will expire. For example, an entry of the form 198909171755 indicates that the principal's entry will expire on September 17, 1989 at 5:55 in the afternoon.

Modification time: The modification field stores the date on which the principal's entry in the Kerberos database was last changed. It is stored in the same format as the expiration time.

Modifier's name: The eleventh field stores the name of the utility that last modified the principal's entry. Only `db_creation` and a blank entry are possible in the modifier's name field. A blank entry indicates that the field was added by `kdb_edit(8krb)`. A modifier name field that states that the entry was produced by `db_creation` indicates that the entry was added by `kdb_init(8krb)` when the database was created.

Modifier's instance: The twelfth field indicates the instance of the utility that last modified the principal's entry. This field is always blank.

Examples

The following is an example of an entry form of a `krb_dbase`-formatted file for host, `cactus`.

```
kprop cactus 255 2 1 0 8f68f19 a941c6d 200001010459 198909171755 * *
```

Files

`/var/dss/kerberos/dbase/dbase`

See Also

ndbm(3), kdb_init(8krb), kdb_edit(8krb), kdb_destroy(8krb), kdb_util(8krb)

Name

krb_slaves – a list of Kerberos slaves

Description

The command `kprop(8krb)` takes as a parameter the name of a file in which a list of Kerberos slave machines is stored. This file must be in `krb_slaves` format.

Each line of a `krb_slaves`-formatted file consists of the machine name of a machine which is running a Kerberos secondary server.

If `cactus.dec.com`, `dopey`, and `walrus.dec.com` run a Kerberos secondary server, the `krb_slaves` file for the Kerberos primary is as follows:

```
cactus.dec.com
dopey
walrus.dec.com
```

See Also

kpropd(8krb), kprop(8krb)

L-devices (5)

Name

L-devices – devices used to connect to remote systems

Syntax

/usr/lib/uucp/L-devices

Description

The `uucp` utility uses the L-devices file. The file contains information about call units and direct connections. It is used to map specifiers in the `L.sys`(5) file to specific devices.

The format of each entry, with each field separated by blanks or tabs, is:

type line call-unit speed brand proto

type A device type, such as ACU or DIR. DIR indicates that this is a direct-connect, hard-wired line.

line The device for the modem line or hard-wired line as named in `/dev`, such as cul0 or ttyab. The special device files are assumed to be in the `/dev` directory.

call-unit The automatic call unit associated with *line*, for example, cua0. Hard-wired lines should place the device for the line in this field, for example, ttyab. The value for *call-unit* is usually the same as the value for *line*.

speed The line speed.

brand The brand name of the modem or ACU. Acceptable brands are DF02, DF03, or DF112 (for DIGITAL modems), Ventel, Hayes, and Vadic. For direct connection, place the word direct in this field.

proto The preferred protocol type to use, for example, `g` or `f`.

Examples

Here are some typical L-devices entries:

```
ACU cua0 cua0 300 DF02
ACU cua1 cua1 1200 DF03
ACU cua2 cua2 1200 DF112
DIR ttyab ttyab 9600 direct
```

See Also

Guide to the uucp Utility

Name

L-dialcodes – dial code abbreviations

Syntax

/usr/lib/uucp/L.dialcodes

Description

The `uucp` utility uses the `L-dialcodes` file. The file contains the dialcodes used in the `L.sys`(5) file (for example, nh, which stands for New Hampshire). The entry format, with the fields separated by blanks or tabs, is:

abb dial-seq

abb The abbreviation used in the `L.sys`(5) file.

dial-seq The dial sequence to call that location.

Examples

The following entry in the `L-dialcodes` file would force any `L.sys` file entry that used the prefix ''nh'' in the phone field to send 603 to the dial unit before the rest of the phone number is dialed:

```
nh 603
```

See Also

Guide to the uucp Utility

L.cmds (5)

Name

L.cmds – allowable remote execution commands

Syntax

/usr/lib/uucp/L.cmds

Description

The `uucp` utility uses the `L.cmds` file to determine which commands can be executed by remote systems with the `uux`(1c) command. The `uucp` utility first looks in the `USERFILE` file to find the execution level defined there for the remote system. Then, `uucp` looks in the `L.cmds` file. The remote system can execute any command whose execution level as defined in `L.cmds` is less than or equal to the execution level for the system as defined in `USERFILE`. The format of the `L.cmds` file is:

command X#

command An ULTRIX system command or application program.

X# The execution level associated with *command*. The number # can range from 0 through 9. If the X field is omitted, then 9 is the default. If X is specified but # is omitted, then 0 is the default, enabling any system to use this command.

You can also specify a line in the `L.cmds` file to define the paths used to search for commands:

PATH=*path1*:*path2*:...

path1 The first directory examined for commands.

path2 The second directory examined for commands.

.
.
.

Examples

This example allows remote systems defined in the local system's `USERFILE` with an execution level of 1 or higher to execute the commands rmail and rnews. Only remote systems with an execution level of 9 would be able to execute `uux`.

```
rmail X1
rnews X1
uux X9
```

See Also

USERFILE(5), uux(1c)
Guide to the uucp Utility

Name

L.sys – information needed to connect to a system

Syntax

/usr/lib/uucp/L.sys

Description

The `uucp` utility uses the `L.sys` file. The file contains entries for each remote system that the local system can call and for each remote system for which the local system accepts calls but does not call. More than one line can be used for a particular remote system. In this case, the additional lines represent alternative communication paths that are tried in sequential order.

The format of each entry, with each field separated by blanks or tabs, is:

system-name time device class phone login

system-name
The name of the remote system.

time A string that indicates the days of the week and the times of day when the system can be called (for example, MoTuTh0800-1740).

The day portion may be a list containing:

Su Mo Tu We Th Fr Sa

The day may also be Wk for any weekday or Any for any day.

You can indicate hours in a range (for example, 0800-1230). If you do not specify a time, calls will be allowed at any time.

Note that a time range that spans 0000 is permitted. For example, 0800-0600 means that all times are allowed except times between 6 AM and 8 AM.

Multiple date specifications that are separated by a vertical bar (|) are allowed. For example, Any0100-0600|Sa|Su means that the system can be called any day between 1 AM and y AM or any time on Saturday and Sunday.

An optional subfield is available to indicate the minimum time, in minutes, before retrying a failed connection. A failed connection attempt is a login failure, as opposed to a dialing connection failure. The subfield separator is a comma (,). For example, Any, 9 means call any time, but wait at least 9 minutes after a failure has occurred.

device Either the ACU or the hard-wired device used for the call. For the hard-wired device, use the last part of the special file name (for example, tty2).

class The line speed for the call (for example, 1200). The exception is when the BC library routine dialout is available, in which case this is the dialout class.

phone The telephone number, made up of an optional alphabetic abbreviation and a numeric part. The abbreviation should be one that appears in the `L-dialcodes` file (for example, ct5900, nh6511). If a numeric number is

used, it should be given in full. For the hard-wired devices, this field contains the same string as used for the *device* field.

login The login information, given as a series of fields and subfields in this format:

expect1 [–[*sendspecial*]–*expect2*] *send* ...

The *expect1* argument is the string the local system expects to read when logging in to the remote system, and the *send* argument is the string the local system is to send when the *expect* string is received. If two double quotation marks ("") are specified instead of the *expect1* argument, nothing is expected from the remote system.

The *sendspecial* argument specifies a special character to be sent to the remote system if the *expect1* argument is not received. If *sendspecial* is omitted, and two dashes (--) follow the *expect1* argument, the local system sends a carriage return to the remote system.

Other special characters are:

EOT	Send an EOT character
BREAK#	Send # break sequences (default is 3)
PAUSE#	Pause # seconds (default is 5)
\d	Pause 1 second before sending next character
\s	Send a blank character
\r	Send a carriage return
\b	Send a break character
\#	Send the character represented by the octal number #. For example, \05 is CTRL/e.
P_ZERO	Change parity from even (default) to zero
P_EVEN	Change parity to even
P_ODD	Change parity to odd
P_ONE	Change parity to one parity

The *expect2* argument defines another string expected to be read after transmission of the *sendspecial* argument to the remote system.

Examples

In the following example, the remote system is expected to send the string ''login:'', to which the local system replies ''xuucp''.

```
login: xuucp ssword: smiley
```

Then the word ''ssword:'' is expected from the remote system. (The first letter of the password prompt varies from system to system, so it is safer to look for the ending characters.) When ''ssword:'' is received, the local system sends ''smiley''. If the login is successful, the conversation between the peer transfer processes (uucico) begins. If the login fails, the connection attempt fails.

In the following example, "login:" is expected.

```
login:--login xuucp ssword: smiley
```

If it is received, "xuucp" is sent to the remote system. If login: is not received, a carriage return is sent to the remote system, and "login:" is expected. If it is received, xuucp is sent to the remote system. The example then proceeds the same as the previous example.

In the following example, "login:" is expected.

```
login:-BREAK1-login: xuucp ssword: smiley
```

If it is not received, one break sequence is sent to change the baud rate of the remote process. Then, "login:" is again expected, and the example proceeds the same as the previous examples.

See Also

L-dialcodes(5)
Guide to the uucp Utility

lang (5int)

Name

lang – language names

Description

The language support databases used by `setlocale` are stored in the directory `/usr/lib/intln`. If either the language support databases are moved or you specify your own language support database, it is necessary to set the INTLINFO environment variable to the new location of these tables. The syntax of this environment variable is identical to `NLSPATH`. See the `environ`(5int) reference page for more information.

Should you want to create your own database, use the `nl_langinfo`(5int) reference page and the *Guide to Developing International Software* as references for what information your database should contain. After you creat the database, you can specify it by using the international compiler, `ic`.

The following table defines the supplied settings of the LANG and LC_ environment variables.

Database	Language	Territory	Codeset	Use
ENG_GB.MCS	English	United Kingdom	DEC MCS	VT200 series
FRE_FR.MCS	French	France		
GER_DE.MCS	German	Germany		
ENG_GB.8859	English	United Kingdom	ISO Latin-1	VT300 series
FRE_FR.8859	French	France		
GER_DE.8859	German	Germany		
ENG_GB.646	English	United Kingdom	ISO 646	VT100 series
FRE_FR.646	French	France		
GER_DE.646	German	Germany		

In addition to the default collation definition for the GER_DE.nnn language, Digital provides a character collation table that collates information using the German telephone directory ordering of data. The following example shows how to set the `LC_COLLATE` variable to use this table with the ISO Latin-1 codeset:

```
LC_COLLATE = GER_DE.8859@P_TELEPHONE
```

See Also

ic(1int), intro(3int), nl_langinfo(3int), setlocale(3int), environ(5int)
Guide to Developing International Software

Name

ldfcn – common object file access routines

Syntax

#include <stdio.h>
#include <filehdr.h>
#include <syms.h>
#include <ldfcn.h>

Description

The common object file access routines are a collection of functions that read an object file that is in common object file form. The calling program must know the detailed structure of the parts of the object file that it processes, but the calling program does not have to know the overall structure of the object file as the routines handle this function.

The interface between the calling program and the object file access routines is based on the defined type `LDFILE` (defined as `struct ldfile`), which is declared in the header file `<ldfcn.h>`. Primarily, this structure provides uniform access to simple object files and object files that are members of an archive file.

The function `ldopen`(3x) allocates and initializes the `LDFILE` structure, reads in the symbol table header, if present, and returns a pointer to the structure to the calling program. The fields of the `LDFILE` structure can be accessed individually through macros defined in `<ldfcn.h>`. The fields contain the following information:

LDFILE *ldptr;

TYPE(ldptr) The file magic number, used to distinguish between archive members and simple object files.

IOPTR(ldptr) The file pointer returned by `fopen`(3s) and used by the standard input/output functions.

OFFSET(ldptr) The file address of the beginning of the object file; if the object file is a member of an archive file, the offset is nonzero.

HEADER(ldptr) The file header structure of the object file.

SYMHEADER(ldptr) The symbolic header structure for the symbol table associated with the object file.

PFD(ldptr) The file table associated with the symbol table.

SYMTAB(ldptr) A pointer to a copy of the symbol table in memory. It is accessed through the pCHDR structure (see `cmplrs/stsupport.h`). If no symbol table is present, this field is NULL. This macro causes the whole symbol table to be read.

LDSWAP(ldptr) If the header and symbol table structures are swapped within the object file and all access requires using `libsex`, this field is set to true. Note that if you use `libmld` routines, all structures, except the optional header and auxiliaries, are swapped.

The object file access functions can be divided into four categories:

(1) Functions that open or close an object file

`ldopen`(3x) and `ldaopen`
: open a common object file

`ldclose`(3x) and `ldaclose`
: close a common object file

(2) Functions that return header or symbol table information

`ldahread`(3x)
: read the archive header of a member of an archive file

`ldfhread`(3x)
: read the file header of a common object file

`ldshread`(3x) and `ldnshread`
: read a section header of a common object file

`ldtbread`(3x)
: read a symbol table entry of a common object file

`ldgetname`(3x)
: retrieve a symbol name from a symbol table entry or from the string table

`ldgetaux`(3x)
: retrieve a pointer into the aux table for the specified ldptr

`ldgetsymstr`(3x)
: create a type string (for example, C declarations) for the specified symbol

`ldgetpd`(3x)
: retrieve a procedure descriptor

`ldgetrfd`(3x)
: retrieve a relative file table entry

(3) Functions that position (seek to) an object file at the start of the section, relocation, or line number information for a particular section

`ldohseek`(3x)
: seek to the optional file header of a common object file

`ldsseek`(3x) and `ldnsseek`
: seek to a section of a common object file

`ldrseek`(3x) and `ldnrseek`
: seek to the relocation information for a section of a common object file

`ldlseek`(3x) and `ldnlseek`
: seek to the line number information for a section of a common object file

`ldtbseek`(3x)
: seek to the symbol table of a common object file

(4) Miscellaneous functions

`ldtbindex`(3x)
: return the index of a particular common object file symbol table entry

`ranhashinit`(3x)
: initialize the tables and constants so that the archive hash and lookup routines can work

`ranhash`(3x)

give a string return the hash index for it

`ranlookup`(3x)
return an archive hash bucket that is empty or matches the string argument

`disassembler`(3x)
print MIPS assembly instructions

`ldreadst`(3x)
cause a section of the symbol table to be read

These functions are described in detail in the manual pages identified for each function.

The `ldopen` and `ldaopen` functions both return pointers to a `LDFILE` structure.

Macros

Additional access to an object file is provided through a set of macros defined in `<ldfcn.h>`. These macros parallel the standard input/output file reading and manipulating functions. They translate a reference of the `LDFILE` structure into a reference to its file descriptor field.

The following macros are provided:

GETC(ldptr)
FGETC(ldptr)
GETW(ldptr)
UNGETC(c, ldptr)
FGETS(s, n, ldptr)
FREADM((char *) ptr, sizeof (*ptr), nitems, ldptr)
FSEEK(ldptr, offset, ptrname)
FTELL(ldptr)
REWIND(ldptr)
FEOF(ldptr)
FERROR(ldptr)
FILENO(ldptr)
SETBUF(ldptr, buf)
STROFFSET(ldptr)

The STROFFSET macro calculates the address of the local symbol's string table in an object file. See the manual entries for the corresponding standard input/output library functions for details on the use of these macros. (The functions are identified as 3s in Section 3 of the reference pages.)

Restrictions

The macro `FSEEK` defined in the header file `<ldfcn.h>` translates into a call to the standard input/output function `fseek`(3s). `FSEEK` should not be used to seek from the end of an archive file since the end of an archive file cannot be the same as the end of one of its object file members.

See Also

ar(1), fopen(3s), fseek(3s), ldahread(3x), ldclose(3x), ldfhread(3x), ldgetname(3x), ldlread(3x), ldlseek(3x), ldohseek(3x), ldopen(3x), ldrseek(3x), ldlseek(3x), ldshread(3x), ldtbindex(3x), ldtbread(3x), ldtbseek(3x)

Name

limits – header files for implementation-specific constants

Syntax

#include *<limits.h>*
#include *<float.h>*

Description

The header file *<limits.h>* specifies the sizes of integral types as required by the proposed ANSI C standard. The header file *<float.h>* specifies the characteristics of floating types as required by the proposed ANSI C standard. The constants that refer to long doubles that should appear in *<float.h>* are not specified because RISC does not implement long doubles.

The file *<limits.h>* contains the following:

```
#define  CHAR_BIT  8          /* # of bits in a 'char'                */
#define  SCHAR_MIN (-128)     /* min integer value of a 'signed       */
                              /* char'                                */
#define  SCHAR_MAX (+127)     /* max integer value of a 'signed       */
                              /* char'                                */
#define  UCHAR_MAX 255        /* max integer value of 'unsigned       */
                              /* char'                                */
#define  CHAR_MIN  (-128)     /* min integer value of a 'char'        */
#define  CHAR_MAX  (+127)     /* max integer value of a 'char'        */
#define  SHRT_MIN  (-32768)   /* min decimal value of a 'short'       */
#define  SHRT_MAX  (+32767)   /* max decimal value of a 'short'       */
#define  USHRT_MAX 65535      /* max decimal value of 'unsigned       */
                              /* short'                               */
#define  INT_MIN   (-2147483648) /* min decimal value of an 'int'     */
#define  INT_MAX   (+2147483647) /* max decimal value of a 'int'      */
#define  UINT_MAX  4294967295  /* max decimal value of 'unsigned      */
                               /* int'                                */
#define  LONG_MIN  (-2147483648) /* min decimal value of a 'long'     */
#define  LONG_MAX  (+2147483647) /* max decimal value of a 'long'     */
#define  ULONG_MAX 4294967295  /* max decimal value of 'unsigned      */
                               /* long'                               */
#define  USI_MAX  4294967295    /* max decimal value of 'unsigned'    */
#define  WORD_BIT   32            /* # of bits in a ''word'' or       */
                                  /* 'int'                            */
#define CHILD_MAX       25       /* max # of processes per user id   */
#define CLK_TCK 60               /* # of clock ticks per second      */
#define LOCK_MAX    0   */ max # of entries in system                 */
                       /* lock table                                 */
#define LINK_MAX        32766    /* max # of links to a single       */
                                 /* file                             */
#define LONG_BIT        32       /* # of bits in a "long" (X/OPEN)   */
#define NAME_MAX        255  /* max # of characters in a file name   */
#define NGROUPS_MAX     32             /* max # of groups            */
#define MAX_INPUT       256      /* max # of bytes in terminal       */
                                 /* input queue                      */
#define MAX_CANON       256      /* max # of bytes in term canon     */
                                 /* input line                       */
#define MAX_CHAR        256      /* max # of bytes in term canon     */
                                 /* input line (X/OPEN)              */
#define OPEN_MAX        64       /* max # of files a process can     */
                                 /* have open                        */
#define PASS_MAX        8        /* max # of characters in           */
```

```
                                    /* a password                    */
#define PATH_MAX        1024        /* max # of characters in        */
                                    /* a path name                   */
#define PID_MAX         30000       /* max value for a process ID    */
#define SYSPID_MAX      2           /* max value for a system        */
                                    /* proc ID (X/OPEN)              */
#define PIPE_BUF        4096        /* max # bytes atomic in         */
                                    /* write to pipe                 */
#define PIPE_MAX        4096        /* max # bytes written to a      */
                                    /* pipe in a write               */

#define PROC_MAX        100     /* max # of simultaneous processes  */
#define STD_BLK         8192    /* # bytes in a physical I/O block  */
#define SYS_NMLN        32 /* # of chars in uname-returned strings */
#define SYS_OPEN        200 /* max # of files open on system        */
#define TMP_MAX 17576 /* max # of calls to tmpnam(3S) before       */
                      /* recycling of names occurs                 */
#define UID_MAX         32000       /* max value for a user or group */
                                    /* ID                            */
#define NZERO 20
                                    /* default nice value (as seen   */
                                   /* by nice(3), not intrinsically) */
/*
 * Internationalization constants
 */
#define MB_LEN_MAX      1   /* max number of bytes in a multibyte   */
                            /* character, any locale: placeholder   */
#define NL_ARGMAX       9   /* max value of digits in calls to      */
                            /* nl_scanf(3S) and nl_printf(3S)       */
#define NL_MSGMAX       32767   /* max message number               */
#define NL_NMAX         2     /* max n-to-1 bytes in mapping chars  */
#define NL_SETMAX       255     /* max set number                   */
#define NL_TEXTMAX      256     /* max no. of bytes in a message    */
                                /* string                           */
#define NL_LBLMAX       32767   /* max number of labels             */
                                /* in catalogue                     */
#define NL_LANGMAX      32      /* max number of bytes in LANG      */
                                /* name                             */
/*
 *      POSIX minimum values
 *
 *      These values are the MINIMUM allowable by POSIX. Actual values
 *      for ULTRIX are defined above.
 */
#define _POSIX_ARG_MAX   4096   /* Length of arguments for          */
                                /* exec( )                          */
#define _POSIX_CHILD_MAX 6      /* Number of simultaneous           */
                                /* procs per uid                    */
#define _POSIX_LINK_MAX  8      /* Number of file links             */
#define _POSIX_MAX_CANON 255    /* Number of bytes in               */
                                /* a terminal canon                 */
                                /* input queue                      */
#define _POSIX_MAX_INPUT 255    /* Number of bytes                  */
                                /* for which space is               */
                                /* guaranteed in terminal           */
                                /* input queue                      */
#define _POSIX_NAME_MAX  14     /* Number of bytes in a filename    */
#define _POSIX_NGROUPS_MAX 0    /* Number of allowable              */
                                /* supplementary group ID's         */
#define _POSIX_OPEN_MAX  16     /* Number of files open at one      */
                                /* time by a given process.         */
#define _POSIX_PATH_MAX  255    /* Number of bytes in a pathname    */
#define _POSIX_PIPE_BUF  512    /* Number of bytes that is          */
```

```
                                        /* guaranteed to be written        */
                                        /* atomically when writing to      */
                                        /* a pipe.                         */
```

The file *<limits.h>* contains the following values for RISC architecture:

```
#define ARG_MAX         20480   /* max length of arguments to exec */
#define HUGE_VAL        1.8e+308 /* infinity                       */
```

The file *<limits.h>* contains the following value for VAX architecture:

```
#define ARG_MAX         10240   /* max length of arguments to exec */
```

The file *<limits.h>* contains the following value for VAX D-float architecture:

```
#define HUGE_VAL        1.70141183460469293e+38
```

The file *<limits.h>* contains the following value for VAX G-float architecture:

```
#define HUGE_VAL        8.9884656743115790e+307
```

The file *<float.h>* contains the following values for RISC architecture:

```
#define  FLT_RADIX  2 /* radix of exponent representation          */
#define  FLT_ROUNDS  1 /* addition rounds                          */
                      /* (>0 implementation-defined)              */
/* number of base-FLT_RADIX digits in the floating point mantissa  */
#define  FLT_MANT_DIG  24
#define  DBL_MANT_DIG  53
/* minimum positive floating-point number x such that              */
/* 1.0 + x < > 1.0                                               */
#define  FLT_EPSILON  1.19209290e-07
#define  DBL_EPSILON  2.2204460492503131e-16
/* number of decimal digits of precision                          */
#define  FLT_DIG  6
#define  DBL_DIG  15
/* minimum negative integer such that FLT_RADIX raised to that    */
/* power minus 1 is a normalized floating point number            */
#define  FLT_MIN_EXP  (-125)
#define  DBL_MIN_EXP  (-1021)
/* minimum normalized positive floating-point number              */
#define  FLT_MIN  1.17549435e-38
#define  DBL_MIN  2.2250738585072014e-308
/* minimum negative integer such that 10 raised to                */
/* that power is in the range of normalized floating-point numbers */
#define  FLT_MIN_10_EXP  (-37)
#define  DBL_MIN_10_EXP  (-307)
/* maximum integer such that FLT_RADIX raised to that             */
/* power minus 1 is a representable finite floating-point number  */
#define  FLT_MAX_EXP  +128
#define  DBL_MAX_EXP  +1024
/* maximum representable finite floating-point number             */
#define  FLT_MAX  3.40282347e+38
#define  DBL_MAX  1.7976931348623157e+308
/* maximum integer such that 10 raised to that power is in        */
/* the range of representable finite floating-point numbers       */
#define  FLT_MAX_10_EXP  38
#define  DBL_MAX_10_EXP  308
```

The file *<float.h>* contains the following values for VAX architecture:

```
#define  FLT_RADIX  2 /* radix of exponent representation          */
#define  FLT_ROUNDS  1 /* addition rounds                          */
```

```
                        /* (>0 implementation-defined)              */
#define FLT_MIN_10_EXP  (-37)
#define DBL_MIN_10_EXP  (-307)
#define LDBL_MIN_10_EXP

#define FLT_MANT_DIG 24
#define FLT_EPSILON 5.96046448e-08
#define FLT_DIG  6
#define FLT_MIN_EXP  (-127)
#define FLT_MIN float  2.93873588e-39
#define FLT_MIN_10_EXP  (-38)
#define FLT_MAX_EXP     127
#define FLT_MAX 1.701411733192644299e+38
#define FLT_MAX_10_EXP  380
```

The file *<float.h>* contains the following values for VAX D-float:

```
#define DBL_MANT_DIG    56
#define DBL_EPSILON     1.3877787807814457e-17
#define DBL_DIG 16
#define DBL_MIN_EXP     (-127)
#define DBL_MIN 2.93873587705571880e-39
#define DBL_MIN_10_EXP  (-38)
#define DBL_MAX_EXP     127
#define DBL_MAX 1.70141183460469293e+38
#define DBL_MAX_10_EXP  38
```

The file *<float.h>* contains the following values for VAX G-float (cc - Mg):

```
#define DBL_MANT_DIG    53
#define DBL_EPSILON     1.1102230246251570e-016
#define DBL_DIG 15
#define DBL_MIN_EXP     (-1023)
#define DBL_MIN 5.56268464626800350e-309
#define DBL_MIN_10_EXP  (-308)
#define DBL_MAX_EXP     1023
#define DBL_MAX 8.9884656743115790e+307
#define DBL_MAX_10_EXP  307
```

linenum (5)

Name

linenum – line number entries in a MIPS object file

Description

If the `cc` command is invoked with the `-g` option, an entry is generated in the object file for each C source line on which a breakpoint can occur. You can then reference line numbers when using the appropriate software test system.

See Also

cc(1), dbx(1), a.out(5)

Name

ltf - labeled tape facility

Description

The term "ltf" (Labeled Tape Facility) refers to the group of programs required to fulfill the features and functionality outlined here.

References for the substance of this document are based on:

1. American National Standard Institute magnetic tape labels and file structure for information interchange ANSI X3.27-1978
2. The document/working paper: Draft Proposed Revision to ANSI X3.27-1978 Public Review Comment on ANSI X3L5/83-28T 15-Oct-84 (describes the version 4 ANSI standard)

This proposed implementation of the `ltf` does not claim to be 100% ANSI standard in all cases. That is, the ULTRIX `ltf` does not support the entirety of the functionality or format capabilities outlined in the documents/publications cited as references.

It should be understood that the functionality and formats for ULTRIX-labeled tapes are simply based on the standards and formats referred to, and described in, the publications/documents listed previously.

The `ltf` attempts to follow these documents as working precepts as accurately as it can, while meeting the needs of ULTRIX systems.

It should be further understood that where the cited documents specify procedures or operational constricts that would conflict with those features/functionality normally found in an ULTRIX system, the procedures or operational constricts, by necessity, have been omitted from the `ltf` implementation.

The goals of `ltf` are to create an accurate exchange of information between ULTRIX systems and between ULTRIX and non-ULTRIX systems as an import/export facility by providing a means to read/write tapes in a format generally acceptable to most systems providing support of ANSI-labeled tapes.

It is not a goal of `ltf` to provide the multivolume file sets or to provide 100% of the ANSI specifications for the following labels on non-ULTRIX generated volumes:

```
VOL1
    Accessibility Field
VOL2 - VOL9
UVL1 - UVL9 (User Volume Labels)
HDR1, EOV1, EOF1
    File-set Identifier
    Expiration Date
    Accessibility Field
    Block Count
HDR2
    Buffer Offset Content
HDR3 - HDR9
UHL1 - UHL9 (User Header Labels)
EOV3 - EOV9 (End of Volume Labels)
EOF3 - EOF9 (End of File Labels)
ULT1 - UTL9 (User Trailer Labels)
```

The FORMATS section provides the general ANSI volume and label formats. Each label consists of 80 bytes of ASCII data as specified. Items enclosed in parentheses () indicate optional fields that may be present according to the ANSI standard. The following terms are used as indicated:

"a" Refers to the sets of characters including uppercase (A-Z), numerals (0-9), and special characters (space ! " % & ' ( ) * + , - _ . / : ; < = > ?).

BLOCK
A group of consecutive bytes of data treated as a unit by the storage medium. Blocks are separated by an interblock gap. A block may contain part of a record, all of the record, or many records.

BLOCK LENGTH
The minimum block size is 18 bytes and the maximum is 20480.

RECORD
A set of related data treated as a unit of information:

TAPE MARK
A control block used as a delimiter.

FORMATS for ANSI VERSION 3/4 - MULTI-FILE / SINGLE VOLUME

```
bot       Beginning of tape marker
VOL1      Volume Label (only 1 permitted for ANSI version 3,
          ANSI version 4 OPTIONALLY permits Volume Labels
          2 through 9)
(UVLn)    OPTIONAL User Volume labels ('n' varies from 1 - 9)
HDR1      First File Header Label
HDR2      Second File Header Label
(HDRn)    OPTIONAL File Header Labels ('n' varies from 3 - 9)
(UHLa)    OPTIONAL User File Header Labels (quantity
          unspecified)
tm        Tape Mark
DATA      data blocks of first file
tm        Tape Mark
EOF1      First End of File Label
EOF2      Second End of File Label
(EOFn)    OPTIONAL End of File Labels ('n' varies from 3 - 9)
(UTLn)    OPTIONAL User Trailer Labels
tm        Tape Mark

HDR1      First File Header Label
HDR2      Second File Header Label
(HDRn)    OPTIONAL File Header Labels ('n' varies from 3 - 9)
(UHLa)    OPTIONAL User File Header Labels (number undefined)
tm        Tape Mark
DATA      data blocks of second file
tm        Tape Mark
EOF1      First End of File Label
EOF2      Second End of File Label
(EOFn)    OPTIONAL End of File Labels ('n' varies from 3 - 9)
(UTLn)    OPTIONAL User Trailer Labels
tm        Tape Mark

HDR1      First File Header Label
  .
  .
  .
tm        Tape Mark
```

```
tm       Tape Mark
```

VOL1 - Label Format - ANSI VERSION 3

The following table identifies the volume and supplies volume security information:

Size of Field	First & Last Byte #	Description or Content
3	1 - 3	"VOL"
1	4	"1"
6	5 - 10	Volume Identifier - user specifiable "a" characters, default = "ULTRIX"
1	11	Accessibility Field - not implemented by ltf
26	12 - 37	Reserved by ANSI (spaces)
14	38 - 51	Owner ID - user defined "a" characters, default = spaces
28	52 - 79	Reserved by ANSI (spaces)
1	80	Label Standard Version, 3 for ANSI Version 3

VOL1 - Label Format - ANSI VERSION 4

The following table identifies the volume and supplies volume security information:

Size of Field	First & Last Byte #	Description or Content
3	1 - 3	"VOL"
1	4	"1"
6	5 - 10	Volume Identifier - user specifiable "a" characters, default = "ULTRIX"
1	11	Accessibility Field - not implemented by ltf
13	12 - 24	Reserved by ANSI (spaces)
13	25 - 37	Implementation ID - "a" characters, ULTRIX default = "DECULTRIXnnnn", where nnnn are digits from 0000 to 9999, identifying the version number of ltf which created volume
14	38 - 51	Owner ID - user defined "a" characters, default = spaces
28	52 - 79	Reserved by ANSI (spaces)
1	80	Label Standard Version, 4 for ANSI Version 4

HDR1 - Label Format - ANSI VERSION 3/4

The following file header label identifies and describes the file. Information in this label is limited to "a" characters only:

Size of Field	First & Last Byte #	Description or Content
3	1 - 3	"HDR"
1	4	"1"
17	5 - 21	File ID - Interchange file name, "a" characters

```
6       22 - 27  File Set ID - "000001" since only one
                 file set on single volume
4       28 - 31  File Section Number - "0001"
4       32 - 35  File Sequence Number - starts at "0001"
                 and increments once for each file on volume
4       36 - 39  Generation Number - "0001"
2       40 - 41  Generation Version Number - "00"
6       42 - 47  Creation Date - Julian date, first character
                 denotes century, " " = 1900, "0" = 2000
6       48 - 53  Expiration Date - Julian date, not implemented
                 by ltf, set to " 99366"
1       54       File Security - " "
6       55 - 60  Block Count - "000000"
13      61 - 73  Implementation ID - same as in VOL1
7       74 - 80  Reserved by ANSI (spaces)
```

HDR2 - Label Format - ANSI VERSION 3/4

File header label describes the record format, maximum record size, and maximum block length of the file. Information in this label is limited to "a" characters, except for the content of bytes 16 through 50 if volume is ANSI version 4. The contents of the field in bytes 16 - 50 are for ULTRIX implementation only; thus, if volume is not ULTRIX, these fields are ignored.

```
Size    First &  Description
of      Last
Field   Byte #   or Content
-----   -------  -----------

3       1 - 3    "HDR"
1       4        "2"
1       5        Record Format - "F" = fixed length records
                                 "D" = variable length records
                                 "S" = segmented records
5       6 - 10   Block Length - default = "02048"
5       11 - 15  Record Length - "F" format, length of each data
                                 record
                                 "D" format, maximum length of a
                                 data record including record
                                 control word
                                 "S" format, maximum length of a
                                 data record not including the
                                 segment control word, if scw =
                                 "00000", maximum record length
                                 may exceed 99999 bytes
6       16 - 21  ULTRIX File Status - st_mode returned from a
                 stat(2) call
4       22 - 25  ULTRIX File Owner ID - uid
4       26 - 29  ULTRIX Owner Group Number - gid
4       30 - 33  ULTRIX Link ID Sequence Number - if file is
                 hard linked, contains the file sequence number
                 of the file this file is linked to
3       34 - 36  ULTRIX True File Type - three character
                 representation of the ULTRIX disk file type
                 (see below)
1       37       Carriage Control - "A" = first byte of record
                 contains FORTRAN carriage control character
                                    "M" = record contains all
                                    required forms control
                                    " " = (space) insert carriage
```

```
                                        return, and line feed between
                                        records
10      38 - 47  ULTRIX File Size - in bytes if known, else
                 spaces
1       48       ULTRIX ltf Header Number - number of last HDR
                 containing the full ULTRIX pathname of the
                 file, digit between 3 and 9
1       49       ULTRIX ltf End of File Header Number -
                 number of last EOF containing the full ULTRIX
                 pathname of the file, digit between 3 and 9,
                 if "0", no path name in EOFs
1       50       ULTRIX Hard Link Flag - necessary when the file
                 linked to has not been put on the volume, thus
                 this flag is used for forward references, "0" =
                 no links or symbolic link, "1" = hard links
2       51 - 52  Buffer Offset - number of bytes of Buffer
                 Offset Field, which is the first record in
                 the data block, if the Buffer Offset is greater
                 than zero, not implemented by ltf and set to "00"
28      53 - 80  ANSI reserved (spaces)
```

The ULTRIX disk file type is described in field 34-36 of HDR2. The following list contains the 3-character representations you can use to specify the disk file type:

```
"adf"     - ASCII data file
"asc"     - ASCII text
"arc"     - Archive
"arl"     - Archive Random Library
"asm"     - Assembly language text
"bin"     - Binary data
"bsp"     - Block special file
"cc "     - 'C' program text
"cmp"     - Compressed text file
"com"     - Command text file
"cpi"     - CPIO file
"csp"     - Character special file
"dir"     - Directory
"eng"     - English text
"exe"     - Executable binary
"for"     - Fortran program source
"fuf"     - Fortran Unformatted File
"nul"     - Null/empty file
"oar"     - Old Archive
"pip"     - Named pipe
"rof"     - roff, nroff, troff, or eqn input text
"soc"     - Socket
"sym"     - Symbolic Link
"???"     - Content of file not determined
```

HDR3 - Label Format - ANSI VERSION 3/4

OPTIONAL file header label presence and content ignored if the volume was not created by an ULTRIX system. Content limited to "a" characters if VOL1 field 80 = "3". Embedded spaces in the path names are not permitted.

Size of Field	First & Last Byte #	Description or Content
3	1 - 3	"HDR"
1	4	"3"
10	5 - 14	ULTRIX standard time of last change to file
10	15 - 24	ULTRIX File Owner Name
20	25 - 44	ULTRIX Hostname
36	45 - 80	ULTRIX File Path Name - first 36 characters

HDR4 through HDR9 - Label Format - ANSI VERSION 3/4

OPTIONAL file header label used by the `ltf` to express some fractional component of the file's complete path name, but presence and content ignored if the volume was not created by an ULTRIX system. Content limited to "a" characters if VOL1 field 80 = "3".

Size of Field	First & Last Byte #	Description or Content
3	1 - 3	"HDR"
1	4	"4" - "9"
76	5 - 80	ULTRIX File Path Name - continuation from previous HDR, left justified and padded with blanks if needed

BUHLa - Label Format - ANSI VERSION 3/4

OPTIONAL User File Header Labels not supported by the `ltf`. They are not output and, if present on an input volume, their presence and content is ignored.

Size of Field	First & Last Byte #	Description or Content
3	1 - 3	"UHL"
1	4	any valid "a" character identifying this label
76	5 - 80	Application Dependent

EOV1 - Label Format - ANSI VERSION 3/4

First End-Of-Volume label that, if read before the first End Of File label (EOF1), indicates that the file is continued on the next volume. For valid hardware and software technical limitations, the ULTRIX `ltf` does not support multivolume file sets and therefore does not output these labels. If present on an input volume, they are interpreted as indicating an error condition, due to the fact that some portion of

the file will not be processed. The fields of this label are identical to the contents of the corresponding fields in the First File Header Label (HDR1), with the exceptions noted below. The following diagrams of EOV labels are intended for reference purposes only.

```
Size    First &  Description
of      Last
Field   Byte #    or Content
-----   -------      -----------

3       1 - 3     "EOV"
1       4         "1"
50      5 - 54    same as corresponding fields in HDR1
6       55 - 60   Block Count - number of blocks in which the
                  file was recorded
20      61 - 80   same as corresponding fields in HDR1
```

EOV2 - Label Format - ANSI VERSION 3/4

The fields of the Second End-Of-Volume label are identical to the contents of the corresponding fields in the Second File Header Label (HDR2), with the exceptions noted. The `ltf` does not support the use of EOV labels. See the previous description for EOV1. OPTIONAL End-Of-Volume labels 3 through 9 (EOV3 - EOV9) are not used by the `ltf`. If present on an input volume, their presence and content are ignored by the `ltf`. See NOTES for EOV1.

```
Size    First &  Description
of      Last
Field   Byte #    or Content
-----   -------   -----------

3       1 - 3     "EOV"
1       4         "2"
76      5 - 80    same as corresponding fields in HDR2
```

EOF1 - Label Format - ANSI VERSION 3/4

The fields of the First End-Of-File label are identical to the contents of the corresponding fields in the First File Header Label (HDR1), with the exceptions noted.

```
Size    First &  Description
of      Last
Field   Byte #    or Content
-----   -------      -----------

3       1 - 3     "EOF"
1       4         "1"
50      5 - 54    same as corresponding fields in HDR1
6       55 - 60   Block Count - number of blocks in which the
                  file was recorded
20      61 - 80   same as corresponding fields in HDR1
```

EOF2 - Label Format - ANSI VERSION 3/4

The fields of the Second End-Of-File label are identical to the contents of the corresponding fields in the Second File Header Label (HDR2).

Size of Field	First & Last Byte #	Description or Content
3	1 - 3	"EOF"
1	4	"2"
76	5 - 80	same as corresponding fields in HDR2

EOF3 through EOF9 - Label Format - ANSI VERSION 3/4

OPTIONAL ANSI end-of-file labels used by the `ltf` to express some fractional component of the file's complete path name. The presence and content of these labels are ignored if the volume was not created by an ULTRIX system. Content limited to "a" characters if VOL1 field 80 = "3".

Size of Field	First & Last Byte #	Description or Content
3	1 - 3	"EOF"
1	4	"3" - "9"
76	5 - 80	ULTRIX File Path Name - continuation from HDR9 and previous EOF, left justified and padded with blanks if needed

UTLa through UTLa - Label Format - ANSI VERSION 3/4

OPTIONAL User File Trailer Labels Set is optional. If present on an input volume, it is ignored by the `ltf`. User File Trailer Labels are not output by the `ltf`. If present, they take the form described. Their use is application dependent but not supported by the `ltf`.

Size of Field	First & Last Byte #	Description or Content
3	1 - 3	"UTL"
1	4	any valid "a" character identifying this label
76	5 - 80	Application Dependent

See Also

ltf(1)

Name

magic – magic file for the file command

Syntax

/usr/lib/file/magic

Description

The magic file is used by the `file` command to identify files that have some sort of *magic number*. A magic number is any numeric or string constant that identifies the file containing the constant.

The `magic` file is formatted as follows:

Byte offset
The byte offset is where `magic` information is found in the file. This is the number of bytes from the beginning of the file to the first byte of the magic number or string. This may, optionally, be preceded by a right angle bracket (>) to indicate a continuation line to supply extra information in the printed message.

Value type
The value type is the type of the information to be found at the specified byte offset. The file data is interpreted as the following valid types:

byte	Unsigned char type
short	Unsigned short type
long	Long type
string	Character (byte) string

Optional operator
Describes how the value specified here should be compared with the data at the desired offset. Valid operator characters are: an equal sign, a right angle bracket, and a left angle bracket (=, >, <). If none is specified, = is assumed.

Value
The value to match. Numeric values may be decimal, octal, or hexadecimal. String values are defined as regular expressions here. The regular expressions used here are extended in two ways from regular expression definition in ed(1).

1. Normally unprintable characters may be escaped with a backslash (\). The special characters \n, \b, \r, and \f are allowed. An octal representation can also be used to insert any desired byte value, except 0. Normally, regular expression cannot handle such character values. Because the backslash is used as an escape character while the regular expression is being read in, normal occurrences of a backslash in a regular expression must be escaped with a second backslash. As an example, \(must be written as \\(and \ must be written as \\
2. Text found in a file can also be inserted in the printed string with the use of the \\% delimiter. All text found between these delimiters is substituted into the print string.

   This regular expression search never terminates until a match is

explicitly found or rejected. The special character \n is a valid character in the patterns. Therefore, the pattern .* should never be used here.

major, minor type
The major and minor file type numbers are not used by the `file`(1) command.

String to print
Any desired text string. Data from the file can be included with the use of continuation lines beginning with a right angle bracket (>). Two types of continuation lines are possible, depending on the sign of the byte offset entry.

If the byte offset is positive, the specified data can be printed in the string when requested with an appropriate `printf`(3) format.

If the offset is a negative number, an internal routine will be called to test if a particular string is necessary and, if so, to return it.

The byte offset number is an index to an internal table of routines available for use. Two such routines are currently defined, both for a.out images:

Byte Offset	Returned String(s)
–1:	["old version 7 style symbol table"]
–2:	["setuid "]["setgid "]["sticky "]

Examples

The following is an example of a script. The second line adds setuid, setgid text, if appropriate:

```
0    string ^#![ ]*\%[^ 0*\%  7,4   %s
>-2 long   0                      7,4   %sscript
```

The following is an example of an executable image:

```
>-1 long   0              12,3   %s
0   short  0413           12,4   demand paged pure
>2  short  02             12,4   POSIX
>2  short  01             12,4   SVID
>-2 long   0              12,4   %sexecutable
>16 long   >0             12,4   not stripped
```

The following is an example of a text file:

```
0    string ^ 1h[0-9][0-9][0-9][0-9][0-9]  7,1  sccsfile
```

Files

`/usr/lib/file/magic`

See Also

file(1)

Name

math – math functions and constants

Syntax

#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	π, the ratio of the circumference of a circle to its diameter.
M_PI_2	$\pi/2$.
M_PI_4	$\pi/4$.
M_1_PI	$1/\pi$.
M_2_PI	$2/\pi$.
M_2_SQRTPI	$2/\sqrt{\pi}$.
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.

See Also

intro(3), values(5)

mh-alias (5mh)

Name

mh-alias – alias file for MH message system

Description

Aliasing allows you to send mail to a person or group of persons without typing their complete mail address. Both your MH personal alias file and the (primary) alias file for mail delivery, /usr/new/lib/mh/MailAliases, process aliases in the same way. You can specify the name of your personal alias file in your .mh_profile.

A line of the alias file can have any of the following formats:

```
alias : address-group
alias ; address-group
< alias-file
```

where:

```
address-group  :=  address-list

                |   "<" file
                |   "=" ULTRIX-group
                |   "+" ULTRIX-group
                |   "*"
address-list   :=  address
                |    address-list, address
```

Continuation lines in alias files end with \ followed by the newline character.

Alias-file and file are ULTRIX file names. ULTRIX-group is a group name (or number) from /etc/group. An address is simply an ULTRIX–style mail address. Throughout this file, case is ignored, except for alias-file names.

If the line starts with a <, then the file named after the < is read for more alias definitions. The reading is done recursively, so a < can occur in the beginning of an alias file with the expected results. If the address-group starts with a <, then the file named after the < is read and its contents are added to the address-list for the alias.

If the address-group starts with an =, then the file /etc/group is consulted for the ULTRIX-group named after the =. Each login name occurring as a member of the group is added to the address list for the alias.

In contrast, if the address-group starts with a +, then the file /etc/group is consulted to determine the group-id of the ULTRIX-group named after the +. Each login name occurring in the /etc/passwd file whose group-id is indicated by this group is added to the address list for the alias.

If the address-group is simply *, then the file /etc/passwd is consulted and all login names with a user-id greater than a given number (usually 200) are added to the address list for the alias.

A trailing * on an alias will match just about anything appropriate, as shown in the

following example:

```
sgroup: fred, fear, freida
fred: frated@UCI
ULTRIX-committee: <ultrix.aliases
staff: =staff
wheels: +wheel
everyone: *
news.*: news
```

On the first line of the example, `sgroup` is defined as an alias for the three names `frated@UCI`, `fear`, and `freida`. On the second line of the example, `fred` is defined as an alias for `frated@UCI`. Next, the definition of `ULTRIX-committee` is given by reading the file `ultrix.aliases` in your MH directory, `staff` is defined as all users who are listed as members of the group `staff` in the `/etc/group` file, and `wheels` is defined as all users whose group-id in `/etc/passwd` is equal to the `wheel` group. Finally, `everyone` is defined as all users with a user-id in `/etc/passwd` greater than 200, and all aliases of the form `news`, which is defined to be anything.

The following stages show how aliases are resolved at posting time.

- A list of all the addresses from the message is built and duplicate addresses are eliminated.
- If the message originated on the local host, then alias resolution is performed for those addresses in the message that have no host specified.
- For each line in the alias file, aliases are compared against all of the existing addresses. If there is a match, the matched `alias` is removed from the address list, and each new address in the address-group is added to the address list, if it is not already on the list. The alias itself is not usually output, rather the address-group that the alias maps to is output instead. If the `alias` is terminated with a semicolon (;), instead of a colon (:), both the `alias` and the address are output in the correct format. This makes replies possible, because in MH, aliases and personal aliases are unknown to the mail transport system.

Because the alias file is read line by line, forward references work; but backward references are not recognized, so there is no recursion.

MH alias files are expanded into the headers of messages posted. This aliasing occurs first, at posting time, without the knowledge of the message transport system. In contrast, once the message transport system is given a message to deliver to a list of addresses, for each address that appears to be local, a system-wide alias file is consulted. These aliases are not expanded into the headers of messages delivered.

To use aliasing in MH, do the following:

1. In your `.mh_profile`, choose a name for your primary alias file, say `aliases`, and add the following three lines:

```
ali: -alias aliases
send: -alias aliases
whom: -alias aliases
```

2. Create the file `aliases` in your MH directory.

3. Start adding aliases to your `aliases` file as appropriate. An alias file must not

reference itself directly, or indirectly through another alias file, using the `<file` construct.

Files

`/usr/new/lib/mh/MailAliases`
Primary alias file

See Also

ali(1mh), send(1mh), whom(1mh), group(5mh), passwd(5mh), mtstailor(5mh), conflict(8mh), post(8mh)

Name

mh-format – format file for MH message system

Description

Several MH commands utilize either a `format` string or a `format` file during their execution. For example, `scan`(1mh) uses a format string that specifies how `scan` should generate the `scan` listing for each message; `repl`(1mh) uses a format file that directs it how to generate the reply to a message, and so on.

This reference page describes how to write new format commands or modify existing ones. You should not attempt this unless you are an experienced MH user.

A format string is similar to a `printf`(3c) string, but uses multiletter escapes. When specifying a string, the usual C backslash characters are honored: `\b`, `\f`, `\n`, `\r`, and `\t`. Continuation lines in format files end with a backslash (\) followed by the newline character.

The interpretation model is based on a simple machine with two registers, `num` and `str`. The former contains an integer value, the latter a string value. When an escape is processed, if it requires an argument, it reads the current value of either `num` or `str`; and, if it returns a value, it writes either `num` or `str`.

Escapes are of three types: `components`, `functions`, and `control`. A component escape is specified as `%{name}`, and is created for each header found in the message being processed. For example, `%{date}` refers to the `Date:` field of the appropriate message. A component escape is always string valued.

A control escape is one of: `%<escape`, `%|`, and `%>`. These correspond to if-then-else constructs: if `escape` is not zero (for integer-valued escapes), or not empty (for string-valued escapes), everything up to `%|` or `%>` (whichever comes first) is interpreted; else, then skip to `%|` or `%>` (whichever comes first) and start interpreting again. A function escape is specified as `%(name)`, and is statically defined. The following table lists the function escapes.

Escape	Argument	Returns	Interpretation
nonzero	integer	integer	num is not zero
zero	integer	integer	num is zero
eq	integer	integer	num == width
ne	integer	integer	num != width
gt	integer	integer	width > num
null	string	integer	str is empty
nonnull	string	integer	str is not empty
msg		integer	Message number
cur		integer	Message is current
size		integer	Size of message
strlen	string	integer	Length of str
me		string	User's mailbox
plus		integer	Add width to num
minus		integer	Subtract num from width
charleft		integer	Space left in output buffer
timenow		integer	Seconds since the epoch

When `str` is a date, these escapes are defined:

Escape	Argument	Returns	Interpretation
sec	string	integer	Seconds of the minute
min	string	integer	Minutes of the day
hour	string	integer	Hours of the day (24 hour clock)
mday	string	integer	Day of the month
mon	string	integer	Month of the year
wday	string	integer	Day of the week (Sunday=0)
year	string	integer	Year of the century
yday	string	integer	Day of the year
dst	string	integer	Daylight savings in effect
zone	string	integer	Timezone
sday	string	integer	Day of the week known 1 for explicit in date 0 for implicit –1 for unknown
clock	string	integer	Seconds since the epoch
rclock	string	integer	Seconds prior to current time
month	string	string	Month of the year
lmonth	string	string	Month of the year (long form)
tzone	string	string	Timezone
day	string	string	Day of the week
weekday	string	string	Day of the week (long)
tws	string	string	Official RFC 822 rendering of the date
pretty	string	string	A more user-friendly rendering
nodate	string		Date was not parseable

When `str` is an address, these escapes are defined:

Escape	Argument	Returns	Interpretation
pers	string	string	Personal name of the address
mbox	string	string	Local part of the address
host	string	string	Domain part of the address
path	string	string	Route part of the address
type	string	integer	Type of host –1 for uucp 0 for local 1 for network 2 for unknown
nohost	string	integer	No host was present in the address
ingrp	string	integer	Address appeared inside a group
gname	string	string	Name of the group (present for first address only)
note	string	string	Commentary text
proper	string	string	Official RFC 822 rendering of the address
friendly	string	string	A more user-friendly rendering
mymbox	string		Address refers to the user's mailbox
formataddr	string		Print str in an address list

The default format string for `scan` follows. This has been divided into several pieces for readability. The first part is:

```
%4(msg)%<(cur)+%| %>%<{replied}-%| %>
```

This means that the message number should be printed in four digits; if the message is the current message, then a + is printed. If the message is not the current message, then a space is printed. If a `Replied:` field is present, a - is printed. If no `Replied:` field is present, then a space is printed. Next:

```
%02(mon{date})/%02(mday{date})
```

The hours and minutes are printed in two digits (zero filled). Next:

```
%<{date} %|*>
```

If no PN Date: field is present, then a * is printed; otherwise, a space. Next:

```
%<(mymbox{from})To:%14(friendly{to})
```

If the message is from me, print `To:` followed by a user-friendly rendering of the first address in the To: field.

```
%|%17(friendly{from})%>
```

If the message is not from me, then the `From:` address is printed. Finally:

```
%{subject}%<{body}<<%{body}%>
```

The subject and initial body are printed preceded by the string <<.

Although this seems complicated, this method is flexible enough to extract individual fields and print them in any format the user desires.

If the `-form` *formatfile* switch is given, `scan` will treat each line in the named file as a format string and act accordingly. This lets the user develop template `scan` listing formats. See `/usr/new/lib/mh/scan.time`, `/usr/new/lib/mh/scan.size`, and `/usr/new/lib/mh/scan.timely` for more details.

See Also

ap(8mh), dp(8mh)

mh-mail (5mh)

Name

mh-mail – message format for MH message system

Description

MH processes messages in a particular format. Although neither Bell nor Berkeley mailers produce message files in the format that MH prefers, MH can read message files in that format.

Each user has a mail drop box that initially receives all messages processed by `post`(8mh).

The `inc`(1mh) command reads from the mail drop box and incorporates the new messages found there into the user's own mail folders (typically *+inbox*). The mail drop box consists of one or more messages.

Messages are expected to consist of lines of text. Graphics and binary data are not handled. No data compression is accepted. All text is in ASCII 7-bit data.

The general memo framework of RFC 822 is used. A message consists of a block of information in a rigid format, followed by general text with no specified format. The rigidly formatted first part of a message is called the message header; the free-format portion is called the body. The header must always exist, but the body is optional. These parts are separated by a blank line or by a line of dashes. The following example shows the standard MH mail header:

```
To:
cc:
Subject:
--------
```

The header is composed of one or more header items. Each header item can be viewed as a single logical line of ASCII characters. If the text of a header item extends across several real lines, the continuation lines are indicated by leading spaces or tabs.

Each header item is called a component and is composed of a keyword or name, along with associated text. The keyword begins at the left margin, cannot contain spaces or tabs, cannot exceed 63 characters (as specified by RFC 822), and is terminated by a colon (:).

The text for most formatted components (such as ''Date:'' and ''Message-Id:'') is produced automatically. The only ones entered by the user are address fields such as ''To:'' and ''cc:''. Internet addresses are assigned mailbox names and host computer specifications. The rough format is ''local@domain'', for example, ''MH@UCI'' or ''MH@UCI-ICSA.ARPA''. Multiple addresses are separated by commas (,). A missing host/domain is assumed to be the local host/domain.

A blank line (or a line of dashes) signals that all following text up to the end of the file is the body of the message. No formatting is expected or enforced within the body.

The following is a list of header components that are considered meaningful to MH programs:

Date:	Added by `post(8)`, contains the date and time of the message's entry into the transport system.
From:	Added by `post(8)`, contains the address of the author or authors (may be more than one if a "Sender:" field is present). Replies are typically directed to addresses in the "Reply-To:" or "From:" field. (The former has precedence, if present.)
Sender:	Added by `post(8)` in the event that the message already has a "From:" line. This line contains the address of the actual sender. Replies are never sent to addresses in the "Sender:" field.
To:	Contains addresses of primary recipients.
cc:	Contains addresses of secondary recipients.
Bcc:	Still more recipients. However, the "Bcc:" line is not copied onto the message as delivered, so these recipients are not listed. MH uses an encapsulation method for blind copies. (See `send(1)`.)
Fcc:	Causes `post(8)` to copy the message into the specified folder for the sender, if the message was successfully given to the transport system.
Message-ID:	A unique message identifier added by `post(8)`, if the `-msgid` flag is set.
Subject:	Sender's commentary. It is displayed by `scan(1)`.
In-Reply-To:	A commentary line added by `repl(1)` when replying to a message.
Resent-Date:	Added when redistributing a message by `post(8)`.
Resent-From:	Added when redistributing a message by `post(8)`.
Resent-To:	New recipients for a message resent by `dist(1)`.
Resent-cc:	Still more recipients. See "cc:" and "Resent-To:".
Resent-Bcc:	Even more recipients. See "Bcc:" and "Resent-To:".
Resent-Fcc:	Copy resent message into a folder. See "Fcc:" and "Resent-To:".
Resent-Message–Id:	A unique identifier appended by `post(8)` if the `-msgid` flag is set. See "Message-Id:" and "Resent-To:".
Resent:	Annotation for `dist(1)` under the `-annotate` option.
Forwarded:	Annotation for `forw(1)` under the `-annotate` option.
Replied:	Annotation for `repl(1)` under the `-annotate` option.

Files

`/usr/spool/mail/$USER`
Location of mail drop

See Also

Standard for the Format of ARPA Internet Text Messages (RFC 822)

mh_profile (5mh)

Name

mh_profile – user customization for MH message system

Syntax

~/.mh_profile

Description

Each user of MH is expected to have a file named `.mh_profile` in his or her home directory. This file contains a set of user parameters used by some or all of the MH family of programs. Each line of the file is in the format:

`profile-component:` *value*

The possible profile components are described in the following table. Only `Path:` is mandatory. The others are optional; some have default values if they are not present. In the notation used below, (profile/context, default: *value*) indicates whether the information is kept in the user's MH profile or MH context and indicates what the default value is.

`Path: Mail`
: Locates MH transactions in directory "Mail".
(profile, no default)

`Context: context`
: Declares the location of the MH context file.
(profile, default: <mh–dir>/context)

`Current-Folder:`
: Keeps track of the current open folder.
(context, default: +inbox)

`Previous-Sequence: pseq`
: Names the sequences which should be defined as the *msgs* or *msg* argument given to the program. If not present, or empty, no sequences are defined. Otherwise, for each name given, the sequence is first zeroed and then each message is added to the sequence.
(profile, no default)

`Sequence-Negation: not`
: Defines the string which, when prefixed to a sequence name, negates that sequence. Hence, `notseen` means all those messages that are not a member of the sequence `seen`.
(profile, no default)

`Unseen-Sequence: unseen`
: Names the sequences which should be defined as those messages recently incorporated by `inc`. `Show` knows to remove messages from this sequence once it thinks they have been seen. If not present, or empty, no sequences are defined. Otherwise, for each name given, the sequence is first zeroed and then each message is added to the sequence.
(profile, no default)

`mh-sequences: .mh_sequences`
The name of the file in each folder which defines public sequences. To disable the use of public sequences, leave the value portion of this entry blank.
(profile, default: .mh_sequences)

`atr-seq-folder: 172 178-181 212`
Keeps track of the private sequence called `seq` in the specified folder.
(context, no default)

`Editor: /usr/new/mh/prompter`
Defines editor to be used by `comp`(1mh), `dist`(1mh), `forw`(1mh), and `repl`(1mh).
(profile, default: prompter)

`Msg-Protect: 600`
Defines octal protection bits for message files. See `chmod`(1) for an explanation of the octal number.
(profile, default: 0600)

`Folder-Protect: 700`
Defines protection bits for folder directories. See chmod(1) for an explanation of the octal number.
(profile, default: 0700)

`program: default switches`
Sets default switches to be used whenever the `MH` program is invoked. For example, you could override the `Editor:` profile component when replying to messages by adding a component such as: `repl: -editor /bin/ed`.
(profile, no defaults)

`-next: nexteditor`
Names "nexteditor" to be the default editor after using "lasteditor". This takes effect at "What now?" level in `comp`, `dist`, `forw`, and `repl`. After editing the draft with "lasteditor", the default editor is set to be "nexteditor". If the user types "edit" without any arguments to "What now?", then "nexteditor" is used.
(profile, no default)

`Folder-Stack: folders`
The contents of the folder-stack for the `folder` command.
(context, no default)

`mhe:`
If present, tells `inc` to compose an MH auditfile in addition to its other tasks.

`Alternate-Mailboxes:mh@uci-750a, bug-mh*`
Tells `repl` and `scan` which addresses are really yours. In this way, `repl` knows which addresses should be included in the reply, and `scan` knows if the message really originated from you. Addresses must be separated by a

comma, and the hostnames listed should be the official hostnames for the mailboxes you indicate, as local nicknames for hosts are not replaced with their official site names. For each address, if a host is not given, then that address on any host is considered to be you. In addition, an asterisk (*) may appear at either or both ends of the mailbox and host to indicate wildcard matching.
(profile, default: your user-id)

`Draft-Folder: drafts`
Indicates a default draft folder for `comp`, `dist`, `forw`, and `repl`; which allows more than one draft message to exist at the same time.
(profile, no default)

`digest-issue-list: 1` Tells `forw` the last issue of the last volume sent for the digest `list`.
(context, no default)

`digest-volume-list: 1`
Tells `forw` the last volume sent for the digest `list`.
(context, no default)

`MailDrop: .mail`
Tells `inc` your maildrop, if different from the default. This is superseded by the `$MAILDROP` envariable.
(profile, default: `/usr/spool/mail/$USER`)

`Signature: "Rand MH System"`
Tells `send` your mail signature. This is superseded by the `$SIGNATURE` envariable. The signature must be enclosed in double quotation marks (" ").

The following profile elements are used whenever an MH program invokes some other program such as `more`(1). The `.mh_profile` can be used to select alternative programs if the user wishes.

The default values are given in the following examples:

```
fileproc:       /usr/new/mh/refile
incproc:        /usr/new/mh/inc
installproc:    /usr/new/lib/mh/install-mh
lproc:          /usr/ucb/more
mailproc:       /usr/new/mh/mhmail
mhlproc:        /usr/new/lib/mh/mhl
moreproc:       /usr/ucb/more
mshproc:        /usr/new/mh/msh
packproc:       /usr/new/mh/packf
postproc:       /usr/new/lib/mh/post
rmmproc:         none
rmfproc:        /usr/new/mh/rmf
sendproc:       /usr/new/mh/send
showproc:       /usr/ucb/more
whatnowproc:    /usr/new/mh/whatnow
whomproc:       /usr/new/mh/whom
```

If you define the environment variable `$MH`, you can specify a profile other than `.mh_profile` to be read by the MH programs that you invoke. If the value of `$MH` is not absolute (that is, does not begin with a slash (/)) it will be presumed to start from the current working directory. This is one of the very few exceptions in MH where nonabsolute pathnames are not considered relative to the user's MH directory.

Similarly, if you define the envariable `$MHCONTEXT`, you can specify a context other than the normal context file (as specified in the MH profile). As always, unless the value of `$MHCONTEXT` is absolute, it will be presumed to start from your MH directory.

MH programs also support other envariables:

`$MAILDROP` : tells `inc` the default maildrop
This supersedes the `MailDrop:` profile entry.

`$SIGNATURE` : tells `send` and `post` your mail signature
This supersedes the `Signature:` profile entry.

`$HOME` : tells all MH programs your home directory

`$TERM` : tells MH your terminal type
The `$TERMCAP` envariable is also consulted. In particular, these two envariables tell `scan` and `mhl` how to clear your terminal and how many columns wide your terminal is. They also tell `mhl` how many lines long your terminal screen is.

Some envariables are set by `MH` programs for `whatnowproc`. These are:

`$editalt:` the alternative message
Set by `dist` and `repl` during edit sessions, so you can read the message being distributed or replied to. The message is also available through a link called `@` in the current directory if your current working directory and the folder the message lives in are on the same UNIX filesystem.

`$mhdraft:` the path to the working draft
Set by `comp`, `dist`, `forw`, and `repl` to tell the `whatnowproc` which file

to ask What now? questions about. In addition, dist, forw, and repl set $mhfolder if appropriate.

$mhaltmsg
: Set by dist and repl to tell the whatnowproc about an alternative message associated with the draft (the message being distributed or replied to).

$mhdist
: Set by dist to tell the whatnowproc that message redistribution is occurring.

$mheditor
: Set to tell the whatnowproc your choice of editor (unless overridden by -noedit).

$mhuse
: May be set by comp.

$mhmessages
$mhannotate
$mhinplace
: Set by dist, forw, and repl if annotations are to occur. The reason for this is that the MH user can select any program as the whatnowproc, including one of the standard shells. As a result, it is not possible to pass information by way of an argument list.

$mhfolder: the folder containing the alternate message
: Set by dist and repl during edit sessions, so you can read other messages in the current folder besides the one being distributed or replied to. The $mhfolder envariable is also set by show, prev, and next for use by mhl.

Context

In previous versions of MH, the current-message value of a writable folder was kept in a file called cur in the folder itself. In mh.3, the .mh_profile contained the current-message values for all folders, regardless of their writability. In all versions of MH since MH.4, the .mh_profile contains only static information, which MH programs will not update. Changes in context are made to the context file kept in the user's MH directory. This includes, but is not limited to, the Current-Folder entry and all private sequence information. Public sequence information is kept in a file called .mh_sequences in each folder.

The .mh_profile may override the path of the context file by specifying a context entry (this must be in lowercase). If the entry is not absolute (does not start with a slash (/)) it is interpreted relative to the user's MH directory. As a result, you can actually have more than one set of private sequences by using different context files.

Restrictions

The shell quoting conventions are not available in the `.mh_profile`. Each token is separated by white space.

There is some question as to what kind of arguments should be placed in the profile as options. In order to provide a clear answer, recall command line semantics of all MH programs: conflicting switches (for example, `-header` and `-noheader`) may occur more than one time on the command line, with the last switch taking effect. Other arguments, such as message sequences, filenames, and folders, are always remembered on the invocation line and are not superseded by following arguments of the same type. Hence, it is safe to place only switches (and their arguments) in the profile.

If you find that an MH program is being invoked again and again with the same arguments, and those arguments are not switches, there are a few possible solutions to this problem.

The first is to create a (symbolic) link in your `$HOME/bin` directory to the MH program of your choice. By giving this link a different name, you can create a new entry in your profile and use an alternate set of defaults for the MH command.

Similarly, you could create a small shell script which calls the MH program of your choice with an alternate set of invocation line switches. Using links and an alternate profile entry is preferable to this solution.

Finally, if you are a `csh` user, you could create an alias for the command of the form:

```
alias cmd cmd arg1 arg2 ...
```

In this way, you can avoid typing lengthy commands to the shell and still give MH commands safely. Remember that some MH commands invoke others and that, in all cases, the profile is read. This means that aliases are disregarded beyond an initial command invocation.

Files

`$HOME/.mh_profile`
: The user profile or $MH rather than the standard profile

`<mh-dir>/context`
: The user context or $CONTEXT rather than the standard context

`<folder>/.mh_sequences`
: Public sequences for <folder>

See Also

chmod(5), mh(1mh), environ(5)

mtstailor (5mh)

Name

mh-tailor – system customization file for MH message system

Description

The file /usr/new/lib/mh/mtstailor defines run-time options for those MH programs which interact in some form with the message transport system. At present, these user programs are: ap, conflict, inc, msgchk, msh, post, rcvdist, and rcvpack.

The options available are listed below, along with default values and a description of their meanings:

localname:
The host name that MH considers local. If not set, depending on the version of ULTRIX you are running, MH will query the system for this value (For example, <whoami.h>, gethostname). This has no equivalent in the MH configuration file.

servers:
A host or list of hosts running sendmail can be specified. When an MH program is run, it uses this entry to search for a central sendmail to connect to. This is particularly useful for workstation users who may not have sendmail running on their workstations.

systemname:
The name of the local host in the uucp domain. If not set, depending on the version of ULTRIX you are running, MH will query the system for this value. This has no equivalent in the MH configuration file.

mmdfldir: /usr/spool/mail
The directory where maildrops are kept. If this is empty, the user's home directory is used. This overrides the "mail" field in the MH configuration file.

mmdflfil:
The name of the maildrop file in the directory where maildrops are kept. If this is empty, the user's login name is used. This overrides the "mail" field in the MH configuration file.

mmdelim1: \001\001\001\001\n
The beginning-of-message delimiter for maildrops.
mmdelim2: \001\001\001\001\n
The end-of-message delimiter for maildrops.
mmailid: 0
If nonzero, then support for MMailids in /etc/passwd is enabled. Basically, the pw_gecos field in the password file is of the form

My Full Name <mailid>

The MH internal routines that deal with user and full names will return "mailid" and "My Full Name", respectively.

lockstyle: 0
: The locking discipline to perform. A value of 0 means to use the `flock` system call, if available. A value of 1 means to use standard *BellMail* locking always (the name of the lock is based on the file name). A value of 2 means to use *MMDF* locking always (the name of the lock is based on device/inode pairs).

lockldir:
: The name of the directory for making locks. If your system does not have the *flock* syscall, then this directory is used when creating locks. If the value is empty, then the directory of the file to be locked is used.

sendmail: `/usr/lib/sendmail`
: The pathname to the `sendmail` program.

maildelivery: `/usr/new/lib/mh/maildelivery`
: The name of the system-wide default `.maildelivery` file. See `slocal`(1mh) for the details.

everyone: 200
: The highest user-id that should not receive mail addressed to everyone.

noshell: path
: If set, indicates that for each user-id greater than the value set for "everyone" and a login shell equivalent to the given value (for example, `/bin/csh`), mail for "everyone" should not be sent to him or her. This option is useful for handling admin, dummy, and guest logins.

The MH message system has a flexible locking system for making locks on files. There are two `mtstailor` variables you should be aware of: `lockstyle` and `lockldir`. The first controls the method of locking; the second says where lock files should be created. The `lockstyle` variable can take on three values: 0, 1, 2.

A value of 0 means to use the `flock` syscall if you are running on 4.2BSD; otherwise, use a locking style of 1. A value of 1 or 2 specifies that a file should be created whose existence means `locked` and whose nonexistence means `unlocked`. A value of 1 says to construct the lockname by appending `.lock` to the name of the file being locked. A value of 2 says to construct the lockname by looking at the device and inode numbers of the file being locked. If the `lockldir` variable is not specified, lock files will be created in the directory where the file being locked resides. Otherwise, lock files will be created in the directory specified by `lockldir`. Prior to installing MH, you should see how locking is done at your site and set the appropriate values.

Files

`/usr/new/lib/mh/mtstailor`
: tailor file

netgroup(5yp)

Name

netgroup – list of network groups

Description

The `netgroup` file defines network-wide groups used for permission checking when doing remote mounts, remote logins, and remote shells. For remote mounts, the information in the `netgroup` file is used to classify machines; for remote logins and remote shells, it is used to classify users. Each line of the `netgroup` file defines a group and has the following format:

groupname member1,...,member_n

Each member is either another group name or a combination of the host name, user name, and domain name.

Any of the three fields can be empty, in which case a wildcard is assumed. For example, to define a group to which everyone belongs, the following entry could appear in the `netgroup` file:

```
universal (,,)
```

Field names that begin with something other than a letter, digit, or underscore (such as "-") work in the opposite way. For example:

```
justmachines  (analytica,-,suez)
justpeople    (-,babbage,suez)
```

The machine *analytica* belongs to the group *justmachines* in the domain *suez,* but no users belong to it. Similarly, the user *babbage* belongs to the group *justpeople* in the domain *suez,* but no machines belong to it.

Network groups are part of the Yellow Pages data base and are accessed through these files:

`/etc/yp/`*domainname*`/netgroup.dir`
`/etc/yp/`*domainname*`/netgroup.pag`
`/etc/yp/`*domainname*`/netgroup.byuser.dir`
`/etc/yp/`*domainname*`/netgroup.byuser.pag`
`/etc/yp/`*domainname*`/netgroup.byhost.dir`
`/etc/yp/`*domainname*`/netgroup.byhost.pag`

These files can be created from `/etc/netgroup` using `makedbm`(8yp).

Files

`/etc/netgroup`
`/etc/yp/`*domainname*`/netgroup.dir`
`/etc/yp/`*domainname*`/netgroup.pag`
`/etc/yp/`*domainname*`/netgroup.byuser.dir`
`/etc/yp/`*domainname*`/netgroup.byuser.pag`
`/etc/yp/`*domainname*`/netgroup.byhost.dir`
`/etc/yp/`*domainname*`/netgroup.byhost.pag`

See Also

getnetgrent(3yp), makedbm(8yp), ypserv(8yp)

netrc(5)

Name

netrc – Berknet information file (.netrc)

Description

The `.netrc` file contains frequently needed options for network commands.

The `.netrc` file uses the following format:

- Each line of the `.netrc` file defines options for a specific machine.
- A line in the `.netrc` file can be either a machine line or a default line.
- Lines appear in the following order: default, default machine name, machine, machine name, and options. Note that the default line must be the first line in the file if it is present.
- Fields on each line are separated by spaces or tabs.

The following are valid options for a machine line:

Option	Parameter	Default
login	name	localname
password	password	(none)
command	command	(none)
write	yes/no	yes
force	yes/no	no
quiet	yes/no	no

See Also

ftp(1c)

Name

networks – network name file

Description

The `networks` file is an ASCII file that contains information regarding the known networks in the DARPA Internet. For each network, a single-line should be present with the following information:

Official network name
Network number
Aliases

Each `network` name is separated from the next by a newline. Items are separated by any number of blanks or tab characters or both. A number sign (#) indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines that search the file. This file is normally created from the official network data base maintained at the Network Information Control Center (NIC), though local changes may be required to bring it up to date regarding unofficial aliases or unknown networks.

The network number may be specified in the conventional dot (.) notation using the `inet_network` routine from the Internet address manipulation library, `inet(3n)`. Network names may contain any printable character other than a field delimiter, newline, or comment character.

The `networks` database may be distributed in a network by a naming service, such as Yellow Pages or BIND/Hesiod. See the *Guide to the Yellow Pages Service* or the *Guide to the BIND/Hesiod Service* for setup information.

Files

`/etc/networks`

See Also

getnetent(3n)
Guide to the BIND/Hesiod Service
Guide to the Yellow Pages Service

nfs (5nfs)

Name

nfs – Network File System

Description

The Network File System (NFS) is a specific file system implemented under the Generic File System Interface, as described in `gfsi`(5).

NFS provides support for sharing ordinary files and directories in a multivendor networking environment. The system administrator for a file server machine makes a file system available for remote access by placing the name of the file system to be shared in an export list. The administrator for a client machine can import a file system from any server machine that has granted access permission to the requesting client machine. A complete exported file system or any subtree of an exported file system can be imported by the client machine. Once imported, users on the client machine can access files in the remote file system as though they were local files.

See Also

getdirentries(2), getmnt(2), mount(2nfs), mount(2), exports(5nfs), fstab(5), gfsi(5), mount(8nfs), showmount(8nfs)

Name

nl_types - language support database data types

Syntax

#include <nl_types.h>

Description

Two international data types, nl_catd and nl_item, are available for you to use in language support databases. These datatypes are defined in the `<nl_types.h>` file.

The nl_catd datatype is used by the message catalog functions, `catopen`, `catgets`, and `catclose`. Variables of this datatype store message catalog descriptors.

The `nl_langinfo` call uses the nl_item data type. Variables of this datatype store data that gives information about the current locale setting. For example, a data item might specify how to format the dates and times for the current locale. The data is stored in the `<langinfo.h>` file.

See Also

intro(3int), catgetmsg(3int), catgets(3int), catopen(3int), nl_langinfo(3int)
Guide to Developing International Software

ntp.conf(5)

Name

ntp.conf – Network Time Protocol configuration file

Description

The `/etc/ntp.conf` file is the configuration file for the Network Time Protocol (NTP) daemon, `ntpd`. This file must be configured on your system before running `ntpd`.

NOTE

Any host names that you specify in the `/etc/ntp.conf` file must have an entry in the `/etc/hosts` file, or an entry in the master `hosts` database, if the database is being served to your system by BIND/Hesiod or Yellow Pages.

The `/etc/ntp.conf` file has four entry formats:

trusting no
This entry guarantees that your system synchronizes only to the NTP servers identified in the peer and server entries specified. Digital recommends that all systems include the `trusting no` entry.

peer *server*
This entry identifies *server* as one of the NTP servers that your system trusts, and from which your system will accept time synchronization. Your system may also provide time synchronization to this server. Servers can be identified by host name or internet address.

NTP servers should be configured with `peer` entries.

server *server*
This entry identifies *server* as one of the NTP servers that your system trusts, and from which your system will accept time synchronization. Your system can not provide time synchronization to this server. Servers can be identified by host name or internet address.

NTP clients should be configured with `server` entries.

peer /dev/null LOCL 1 -5 local
This entry identifies your system as a local reference clock. A local reference clock is the most accurate system clock available at your site. If you receive time synchronization from the Internet NTP service, you should not include this entry on any of your systems. At most, one system in a set of nodes running `ntpd` should be identified as a local reference clock.

A host which specifies this entry should not specify any `peer` or `server` entries.

Examples

This is a sample configuration file for an NTP client which receives time synchronization from the NTP servers: `server1`, `server2`, and `server3`. Lines beginning with a number sign (#) are comments.

```
#
#               NTP  Configuration File
```

```
#        This file is mandatory for the ntpd daemon
#
#
#
#   **  A L L  **
#
#  "trusting no" prevents this host from synchronizing
#   to any host that is not listed below.  It is recommended
#   that all hosts include the line "trusting no".
#
trusting no
#
#
#    **  S E R V E R  **
#
#  If you are configuring a server, use "peer" entries to
#  synchronize to other NTP servers.  For example, server1,
#  server2, and server3.
#
#peer        server1
#peer        server2
#peer        server3
#
#
#
#
#    **  C L I E N T  **
#
#  If you are configuring a client, use "server" entries to
#  synchronize to NTP servers.  For example, server1, server2,
#  and server3.
#
server       server1
server       server2
server       server3
#
#
#
#    **  L O C A L   R E F E R E N C E   C L O C K  **
#
#  If you are configuring a local reference clock, include the
#  following entry and the "trusting no" entry ONLY.
#
#peer  /dev/null    LOCL  1      -5     local
#
```

See Also

ntp(1), ntpd(8), ntpdc(8)
RFC 1129—Internet time synchronization: The Network Time Protocol
Introduction to Networking and Distributed System Services

passwd(5)

Name

passwd – password file

Description

The `passwd` file is an ASCII file that contains the following information for each user:

Login name
Password field
User ID
Group ID
User's real name, office, extension, home phone
Initial working directory
Program to use as Shell

Each line in the `passwd` file represents a user entry. Each field within a user entry is separated from the next by a colon. Each user entry is separated from the next by a new line. If the password field is null, no password is demanded; if the Shell field is null, then `/bin/sh` is used.

This file resides in directory `/etc`. Because the password, if present, is encrypted, the `passwd` file has general read permission and can be used, for example, to map user IDs to names.

The "user's real name" can contain an ampersand (&), meaning insert the login name. This information is set by the `chfn`(1) command and used by the `finger`(1) command.

Appropriate precautions must be taken to lock the file against changes if it is to be edited with a text editor. The `vipw` command does the necessary locking.

If the system is running UPGRADE security level and an asterisk appears in the password field, or if the system is running ENHANCED level, then the content of the password field is ignored and the password for the account is located in the authentication database. See `auth`(5) and the *Security Guide for Administrators* for more information.

The files `/etc/passwd.dir` and `/etc/passwd.pag`, if they exist, contain the hashed version of the `/etc/passwd` file. If present, they are used by the `getpwnam` and `getpwuid` functions to retrieve entries rapidly. See `mkpasswd`(8) for more information.

The `passwd` database can be distributed in a network by a naming service, such as Yellow Pages or BIND/Hesiod. See the *Guide to the Yellow Pages Service* or the *Guide to the BIND/Hesiod Service* for setup information.

Restrictions

The "name" can contain only lowercase ASCII characters a to z and the numbers 0 to 9.

Files

```
/etc/passwd
/etc/passwd.dir
/etc/passwd.pag
```

See Also

chfn(1), finger(1), login(1), passwd(1), crypt(3), getpwent(3), auth(5), group(5), adduser(8), mkpasswd(8), vipw(8)
Guide to the BIND/Hesiod Service
Guide to the Yellow Pages Service
Security Guide for Administrators

passwd (5yp)

Name

passwd – password file description with the Yellow Pages service implemented

Description

The `passwd` file stores initial login information, including passwords for each user in the system. Regardless of whether or not the system has the Yellow Pages service implemented, the `passwd` file contains the following information:

Name (login name, contains no uppercase)
Encrypted password
Numerical user ID
Numerical group ID
User's real name, office, extension, home phone.
Initial working directory
Program to use as Shell

The name can contain an ampersand (&), meaning insert the login name. This information is set by the `chfn`(1) command and used by the `finger`(1) command.

This is an ASCII file. Each field within each user's entry is separated from the next by a colon. Each user is separated from the next by a new line. If the password field is null, no password is demanded; if the shell field is null, the system defaults to the `/bin/sh` shell.

This file resides in the `/etc` directory. Because of the encrypted passwords, it can and does have general read permission and can be used, for example, to map numerical user IDs to names.

Appropriate precautions must be taken to lock the `/etc/passwd` file against simultaneous changes if it is to be edited with a text editor. The `vipw` command does the necessary locking.

In a Yellow Pages environment, the `passwd` file can also have a line beginning with a plus (+), which means to incorporate entries from the Yellow Pages data base. There are three styles of + entries: by itself, + means to insert the entire contents of the Yellow Pages password file at that point; *+name* means to insert the entry (if any) for *name* from the Yellow Pages at that point; *+@name* means to insert the entries for all members of the network group *name* at that point. If a + entry has a nonnull password, directory, gecos, or shell field, it will override what is contained in the Yellow Pages. The numerical user ID and group ID fields cannot be overridden.

Examples

Here is a sample `/etc/passwd` file:

```
root:q.mJzTnu8icF.:0:10:Privileged Account:/:/bin/csh
jcj:6k/7KCFRPNVXg:508:10:JC Javert:/usr2/jcj:/bin/csh
+john:
+@documentation:no-login:
+:::Guest
```

In this example, there are specific entries for users *root* and *jcj,* in case the Yellow Pages are temporarily out of service. Alternatively, a user may need specific login information on a given system that differs from the information contained in the Yellow Pages map for that user. The user, john, will have his password entry in the

Yellow Pages incorporated without change. Anyone in the netgroup *documentation* will have their password field disabled, and anyone else will be able to log in with their usual password, shell, and home directory, but with a gecos field of *Guest*.

Files

`/etc/passwd`

See Also

chfn(1), finger(1), login(1), passwd(1), crypt(3), getpwent(3), group(5), adduser(8), vipw(8)

patterns (5int)

Name

patterns – patterns for use with internationalization tools

Syntax

See the Description section.

Description

The patterns file contains the patterns that must be matched for the internationalization tools `extract`, `strextract`, and `strmerge`.

The pattern file in the following example is the default patterns file located in `/usr/lib/intln/patterns`.

```
# This is the header to insert at the beginning of the first new
# source file

$SRCHEAD1                                                    (1)
\#include <nl_types.h>
nl_catd _m_catd;
\

# The header to insert at the beginning of the rest of the new
# source files

$SRCHEAD2                                                    (2)
\#include <nl_types.h>
extern nl_catd _m_catd;
\

# This is the header to insert at the beginning of the message
# catalogues

$CATHEAD                                                     (3)
\$ /*
\$  * X/OPEN message catalogue
\$  */
\
\$quote "

# This is how patterns that are matched will get rewritten.

$REWRITE                                                     (4)
catgets(_m_catd, %s, %n, %t)

# Following is a list of the sort of strings we are looking for.
# The regular expression syntax is based on regex(3).

$MATCH                                                       (5)

# Match on strings containing an escaped "
"[^\\]*\\"[^"]*"

# Match on general strings
"[^"]*"

# Now reject some special C constructs.

$REJECT                                                      (6)
```

```
# the empty string
""0

# string with just one format descriptor
"%."
"%.\."

# string with just line control in
"\\."

# string with just line control and one format descriptor in
"%.\\."
"\\.%."

# ignore cpp include lines
\#[  ]*include[  ]*".*"
\#[  ]*ident[  ]*".*"

# reject some common C functions and expressions with quoted
# strings
[sS][cC][cC][sS][iI][dD]\[\][  ]*=[  ]*".*"
open[  ]*([^,]*,[^)]*)
creat[  ]*([^,]*,[^)]*)
access[  ]*([^,]*,[^)]*)
chdir[  ]*([^,]*,[^)]*)
chmod[  ]*([^,]*,[^)]*)
chown[  ]*([^,]*,[^)]*)

# Reject any strings in single line comments
/\*.*\*/

# Print a warning for initialised strings.

$ERROR initialised strings cannot be replaced        (7)
char[^=]*=[  ]*"[^"]*"
char[^=]*=[  ]*"[^\\]*\\"[^"]*"
char[ ]*\**[A-Za-z][A-Za-z0-9]*\[[^\]*\][ ]*=[  {]*"[^"]*"
char[ ]*\**[A-Za-z][A-Za-z0-9]*\[[^\]*\][ ]*=[  {]*"[^\\]*\\"[^"]*"
```

The default patterns file is divided into the following sections:

(1) In the $SRCHEAD1 section, the `strmerge` and `extract` commands place text in this section at the beginning of the first new source program, which is prefixed by `nl_`. These commands define the native language file descriptors that point to the message catalog.

(2) In the $SRCHEAD2 section, the `strmerge` and `extract` commands place text in this section at the beginning of the second and remaining source programs. These commands also define the native language file descriptors that point to the message catalog. $SRCHEAD2 contains the external declaration of the nl file descriptor.

(3) In the $CATHEAD section, the `strmerge` and `extract` commands place text in this section at the beginning of the message catalog.

(4) In the $REWRITE section, you specify how the `strmerge` and `extract` commands should replace the extracted strings in the new source program. You can supply three options to the `catgets` command:

%s This option increments the set number for each source. This option applies only if you are using the `strmerge` command. For more

information on set numbers, see the `catgets`(3int) reference page.

%n This option increments the message number for each string extracted. This option applies if you are using either the `strmerge` or `extract` commands.

%t This option expands the text from the string extracted. The string can be a error message or the default string extracted and printed by the `catgets` command. For example, if you want an error message to appear when `catgets` is unable to retrieve the message from the message catalog, you would include the following line:

```
catgets(_m_catd, %s, %n, "BAD STRING")
```

When `catgets` fails, it returns the message BAD STRING.

(5) In the $MATCH section, you specify the patterns in the form of a regular expression that you want the `strextract`, `strmerge`, and `extract` commands to find and match. The regular expression follows the same syntax rules as defined in `regex`(3) reference page.

(6) In the $REJECT section, you specify the matched strings that you do not want the `strmerge` and `extract` commands to replace in your source program. The regular expression follows the same syntax rules as defined in `regex`(3) reference page.

(7) In the $ERROR section, the `strextract`, `strmerge`, and `extract` commands look for bad matches and notify you with a warning message. The regular expression follows the same syntax rules as defined in the `regex`(3) reference page.

See Also

intro(3int), extract(1int), strextract(1int), strmerge(1int), trans(1int), regex(3)
Guide to Developing International Software

Name

phones – remote host phone number data base

Description

The file `/etc/phones` contains the system-wide private phone numbers for the `tip`(1c) program. This file is normally unreadable, and so can contain privileged information.

The format of the file is a series of lines of the form: <system-name>[\t]*<phone-number>. The system name is one of those defined in the `remote`(5) file and the phone number is constructed from [0123456789-=*%]. The equal sign (=) and the asterisk (*) characters are indicators to the autocall units to pause and wait for a second dial tone when going through an exchange. The equal sign (=) is required by the DF02-AC; the asterisk (*) is required by the BIZCOMP 1030.

Only one phone number per line is permitted. However, if more than one line in the file contains the same system name, `tip`(1c) attempts to dial each one in turn, until it establishes a connection.

Files

`/etc/phones`

See Also

tip(1c), remote(5)

plot(5)

Name

plot – graphics interface

Description

Files in this format are produced by the routines described in `plot`(3x) and are interpreted for various devices by commands described in `plot`(1g).

A graphics file is a stream of plotting instructions. Each instruction consists of an ASCII letter, usually followed by bytes of binary information, executed in order. A point is designated by 4 bytes representing the x and y values and each value is a signed integer. The last designated point in an `l`, `m`, `n`, or `p` instruction becomes the current point for the next instruction.

In the following descriptions, the name of the corresponding routine in `plot`(3x) is enclosed in parenthesis:

a (arc) The first 4 bytes are the center, the next 4 provide the starting point, and the last 4 bytes designate the end point of a circular arc. The least significant coordinate of the end point is used only to determine the quadrant. The arc is drawn counter-clockwise.

c (circle) The first 4 bytes provide the center of the circle and the next 2 bytes designate the radius.

e (erase) Start another frame of output.

f (linemod) Take the following string, up to a new line, as the style for drawing further lines. The styles are dotted, solid, longdashed, shortdashed, and dotdashed. This is only effective in the following plots: *4014, ver, lvp16,* and *hp7475a.*

l (line) Draw a line from the point designated by the next 4 bytes to the point provided by the following 4 bytes.

m (move) The next 4 bytes provide a new current point.

n (cont) Draw a line from the current point to the point designated by the next 4 bytes. For further information, see `plot`(1g).

p (point) Plot the point provided by the next 4 bytes.

s (space) The next 4 bytes give the lower left corner of the plotting area. The following 4 bytes give the upper right corner. The plot is magnified or reduced to fit the device as closely as possible.

Space settings that fill the plotting area with unity scaling are listed below for devices supported by the filters of `plot`(1g). In each of the following cases, the plotting area is assumed square; points outside the square can be displayed on devices that have areas which are not square:

4014	space(0, 0, 3120, 3120);
ver	space(0, 0, 2048, 2048);
300, 300s	space(0, 0, 4096, 4096);
450	space(0, 0, 4096, 4096);
aed	space(0, 0, 511, 482)
bitgraph	space(0, 0, 768, 1024)

dumb	space(0, 0, 132, 90)
gigi	space(0, 0, 767, 479)
grn	space(0, 0, 512, 512)
hp7221	space(0, 0, 1800, 1800)
lvp16	space(0, 0, 10365, 7962) (Paper Size: MET A) space(0, 0, 16640, 10365) (Paper Size: MET B) space(0, 0, 11040, 7721) (Paper Size: US A4) space(0, 0, 16150, 11040) (Paper Size: US A3) space(0, 0, 7721, 7721) (Default)
hp7475a	Same as for lvp16.

t (label) Place the first character of the following ASCII string on the current point. This string is terminated by a newline character.

See Also

graph(1g), plot(1g), plot(3x)

printcap (5)

Name

printcap – printer capability data base

Syntax

/etc/printcap

Description

The `printcap` file describes the printers available on a system. There is one entry in the file for each printer, and the entry describes the printer capabilities. A change to the `printcap` file immediately affects the spooling system, unless the affected queue is active. In this case, the spooling queue should be stopped and restarted. For more information, refer to `lpc(8)`.

Entries in the `printcap` file comprise a number of fields separated by colons (:). The first entry for each printer gives the names that are known for the printer. The names are separated by the pipe character (|). The first name is the name of the printer that will be displayed when you use the `lpc` command to show the status of a queue. Second and subsequent names are alternative names for the printer. You can use the last name to fully identify the printer, including blanks for readability if necessary.

The `/etc/printcap` file is created when the system is installed. After this, you can modify the `printcap` file by using the `lprsetup` script or a suitable editor. The `lprsetup` script is described on the `lprsetup(8)` reference page and in the *Guide to System Environment Setup*.

When a file is printed using the `lpr` command, the file can be sent to a named printer. If a printer is not named, and a print name is not defined by the PRINTER environment variable, the file is sent to the printer with the name "lp" in the `printcap` file. The `printcap` file should always have a printer with the name "lp".

Examples

A typical entry for a printer in the `printcap` file would be:

```
lp|lp0|nlp|ln03 in room 4:\
       :af=/usr/adm/lpacct:\
       :br#4800:\
       :fc#0177777:\
       :fs#03:\
       :if=/usr/lib/lpdfilters/ln03of:\
       :lf=/usr/adm/lperr:\
       :lp=/dev/tty00:\
       :mx#0:\
       :of=/usr/lib/lpdfilters/ln03of:\
       :pl#66:\
       :pw#80:\
       :sd=/usr/spool/lpd:\
       :xc#0177777:\
       :xs#044000:
```

This example shows the format of an entry created using the `lprsetup` script. For more information, refer to the *Guide to System Environment Setup*.

Capabilities

There are three types of capabilities in the `printcap` file: Boolean, string, and numeric. String valued capabilities are processed before use. For more details, refer to `termcap(5)`. The following list contains the names of capabilities that can be used in the `printcap` file:

Name	Type	Default	Description
af	str	NULL	Accounting file name
br	num	none	Baud rate, set if lp is a tty (ioctl call)
cf	str	NULL	Cifplot data filter
ct	str	"dev"	Connection type - only valid when **uv=psv1.0** (choices are: dev, lat,remote, network)
db	num	0	Debugging level (choices are: 0 (none), 1 (normal), 10 (do not execute job, describe actions to log file))
df	str	NULL	Text data filter (DVI format)
du	num		Daemon user id
fc	num	0	If lp is a tty, clear octal flag values (`tty(4)` sg_flags)
ff	str	"\f"	String to send for a form feed
fo	bool	false	Print a form feed when device is opened
fs	num	0	If lp is a tty, set octal flag values (`tty(4)` sg_flags)
gf	str	NULL	Graph data filter (plot(3X) format)
if	str	NULL	Text filter that does accounting
lf	str	"/dev/console"	Error logging file name
lo	str	"lock"	Lock file name
lp	str	"/dev/lp"	Device name to open for output
mc	num		Maximum number of copies allowed
mx	num	1000	Maximum file size (in 1kbyte blocks), 0 = unlimited
nf	str	NULL	Ditroff (device independent troff) data filter
of	str	NULL	Output filtering program name
op	str	NULL	The entry in the ''Name'' field for LAT port characteristics
os	str	NULL	Service name supported on some terminal servers
pl	num	66	Page length (in lines)
pp	str	"/bin/pr"	Print filter
ps	str	"non_PS"	Printer type (choices are: non_PS, LPS)
pw	num	132	Page width (in characters)
px	num	0	Page width in pixels
py	num	0	Page length in pixels

rf	str	NULL	Filter for printing Fortran style text files
rm	str	NULL	Machine name for remote printer
rp	str	"lp"	Remote printer name argument
rs	bool	false	Restrict remote users to those with local accounts
rw	bool	false	Open the printer device for reading as well as writing
sb	bool	false	Short banner (one line only)
sc	bool	false	Suppress multiple copies
sd	str	"/usr/spool/lpd"	Spool directory
sf	bool	false	Suppress form feeds
sh	bool	false	Suppress printing of banner page header
st	str	"status"	Status file name
tf	str	NULL	Troff data filter (cat phototypesetter)
tr	str	NULL	Trailer string to print when queue empties
ts	str	NULL	LAT terminal server node name
uv	str	"3.0"	ULTRIX version number (choices are: 3.0, psv1.0)
vf	str	NULL	Raster image filter
xc	num	0	If lp is a tty, clear local mode octal values (`tty(4)` "Local mode")
xf	str	NULL	Transparent mode filter
xs	num	0	If lp is a tty, set local mode octal values (`tty(4)` "Local mode")

The following capabilities set defaults for PostScript (TM) printers. You should refer to the `lpr(1)` reference page for the choices available for each capability. The equivalent `lpr` options are shown for reference purposes.

Name	Type	Default	Description	lpr Option
Da	str	"ps"	Data type	–D
It	str	NULL	Input tray	–I
Lu	str	NULL	Layup definition file	–L
Ml	str	NULL	Record messages	–M
Nu	str	NULL	Number up	–N
Or	str	"portrait"	Orientation	–O
Ot	str	NULL	Output tray	–o
Ps	str	NULL	Page size	–F
Sd	str	NULL	Default sheet size (see below)	
Si	str	NULL	Sides	–K
Ss	str	NULL	Sheet size	–S
Ul	str	last page	Upper page limit	–Z
Xf	str	"xlator_call"	Translator dispatch program	
Lf	str	"layup"	Layup to PostScript (TM) translator	

Dl	str	"/usr/lib/ lpdfilters/ lps_v3.a"	Name of the device control module library file

The **Ss** capability specifies a mandatory sheet size. The print job fails if this sheet size is not available with the printer.

The **Sd** capability specifies a preferred sheet size and is overridden by the Ss capability and the `lpr -S` command. If the sheet size specified by the `Sd` capability is not available, the print job does not fail, but is printed on the default sheet size for the printer.

The **Dl** capability specifies the name of the device control module library file. This file should be either `/usr/lib/lpdfilters/lps40.a` if the PrintServer supporting host software is Version V2.0 or V2.1, or `/usr/lib/lpdfilters/lps_v3.a` if the PrintServer supporting host software is Version V3.0.

See Also

lpq(1), lpr(1), lprm(1), termcap(5), lpc(8), lpd(8), pac(8)
Guide to System Environment Setup

Name

prof – profile within a function

Syntax

#define MARK
#include <prof.h>

void MARK (name)

Description

The symbol MARK produces a mark called *name* that is treated the same as a function entry point. Execution of the mark increments the counter for that mark, and the program-counter time spent is accounted to the preceding mark or to the function if a preceding mark is not within the active function.

The *name* argument can be any combination of numbers or underscores. Each *name* in a single compilation must be unique, but can 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. For example:

```
cc -p -DMARK foo.c
```

If MARK is not defined, the MARK(name) statements may remain in the source files, but they will be ignored.

Examples

In the following example, marks are used to determine how much time is spent in each loop. Unless the 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)

protocols(5)

Name

protocols – protocol name file

Description

The `protocols` file is an ASCII file that contains information regarding the known protocols used in the DARPA Internet. For each protocol, a single line should be present with the following information:

Official protocol name
Protocol number
Aliases

Each `protocol` name is separated from the next by a new line. Items are separated by any number of blanks or tab characters or both. A number sign (#) indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines that search the file.

Protocol names can contain any printable character other than a field delimiter, newline, or comment character.

The `protocols` database may be distributed in a network by a naming service, such as Yellow Pages or BIND/Hesiod. See the *Guide to the Yellow Pages Services* or the chapter on Hesiod in the *Guide to the BIND/Hesiod Service* for setup information.

Files

`/etc/protocols`

See Also

getprotoent(3n)
Guide to the BIND/Hesiod Service
Guide to the Yellow Pages Service

Name

reloc – relocation information for a MIPS object file

Syntax

#include <reloc.h>

Description

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 */
   unsigned  r_symndx:24,       /* index into symbol table */
             r_reserved:3,
             r_type:4,          /* relocation type */
             r_extern:1;        /* if 1 symndx is an index into the
                                   external symbol table, else symndx
                                   is a section # */
} ;

/* Relocation types */

#define  R_ABS       0
#define  R_REFHALF   1
#define  R_REFWORD   2
#define  R_JMPADDR   3
#define  R_REFHI     4
#define  R_REFLO     5
#define  R_GPREL     6
#define  R_LITERAL   7

/* Section numbers */

#define  R_SN_NULL   0
#define  R_SN_TEXT   1
#define  R_SN_RDATA  2
#define  R_SN_DATA   3
#define  R_SN_SDATA  4
#define  R_SN_SBSS   5
#define  R_SN_BSS    6
#define  R_SN_INIT   7
#define  R_SN_LIT8   8
#define  R_SN_LIT4   9
```

The link editor reads each input section and performs relocation. The relocation entries direct how references found within the input section are treated.

If `r_extern` is zero, it is a local relocation entry and then `r_symndex` is a section number (R_SN_*). For these entries, the starting address for the section referenced by the section number is used in place of an external symbol table entry's value. The assembler and loader always use local relocation entries if the item to be relocated is defined in the object file.

For every external relocation (except R_ABS) a signed constant is added to the symbol's virtual address that the relocation entry refers to. This constant is assembled at the address being relocated.

R_ABS	The reference is absolute and no relocation is necessary. The entry will be ignored.
R_REFHALF	A 16-bit reference to the symbol's virtual address.
R_REFWORD	A 32-bit reference to the symbol's virtual address.
R_JMPADDR	A 26-bit jump instruction reference to the symbol's virtual address.
R_REFHI	A reference to the high 16 bits of the symbol's virtual address. The next relocation entry must be the corresponding R_REFLO entry, so the proper value of the constant to be added to the symbol's virtual address can be reconstructed.
R_REFLO	A reference to low 16 bits to the symbol's virtual address.
R_GPREL	A 16-bit offset to the symbol's virtual address from the global pointer register.
R_LITERAL	A 16-bit offset to the literal's virtual address from the global pointer register.

Relocation entries are generated automatically by the assembler and automatically used by the link editor. Link editor options exist for both preserving and removing the relocation entries from object files.

The number of relocation entries for a section is found in the `s_nreloc` field of the section header. This field is a C language short and can overflow with large objects. If this field overflows, the section header `s_flags` field has the S_NRELOC_OVFL bit set. In this case, the true number of relocation entries is found in the `r_vaddr` field of the first relocation entry for that section. That relocation entry has a type of R_ABS, so it is ignored when the relocation takes place.

See Also

as(1), ld(1), a.out(5), syms(5), scnhdr(5)

Name

remote – remote host description file

Description

The systems known by `tip`(1c) and their attributes are stored in an ASCII file that is structured somewhat like the `termcap`(5) file. Each line in the file provides a description for a single *system*. Fields are separated by colons (:). Lines ending in a backslash (\) followed immediately by a newline character are continued on the next line.

The first entry is the names of the host system. If there is more than one name for a system, the names are separated by vertical bars. After the name of the system comes the fields of the description. A field name followed by an equal sign (=) indicates a string value follows. A field name followed by a number sign (#) indicates a following numeric value.

Entries named "tip*" and "cu*" are used as default entries by `tip` and the `cu` interface to `tip`, as follows. When `tip` is invoked with only a phone number, it looks for an entry of the form "tip300", where 300 is the baud rate with which the connection is to be made. When the `cu` interface is used, entries of the form "cu300" are used.

Capabilities

Capabilities are either strings (str), numbers (num), or Boolean flags (bool). A string capability is specified by *capability=value*; for example, `dv=/dev/harris`. A numeric capability is specified by *capability#value*; for example, `xa#99`. A Boolean capability is specified by simply listing the capability.

at (str) Autocall unit type. This string is what is searched for in `/etc/acucap` to decide if the generic dialer is to be used. For further information, see `acucap`(5).

br (num) The baud rate used in establishing a connection to the remote host. This is a decimal number. The default baud rate is 300 baud.

cm (str) An initial connection message to be sent to the remote host. For example, if a host is reached through port selector, this might be set to the appropriate sequence required to switch to the host.

cu (str) Call unit if making a phone call. Default is the same as the `dv` field.

di (str) Disconnect message sent to the host when a disconnect is requested by the user.

du (bool) This host is on a dial-up line.

dv (str) UNIX devices to open to establish a connection. If this file refers to a terminal line, `tip`(1c) attempts to perform an exclusive open on the device to ensure only one user at a time has access to the port.

el (str) Characters marking an end-of-line. The default is NULL. Tilde (~) escapes are recognized by `tip` only after one of the characters in `el`, or after a carriage-return.

fs (str) Frame size for transfers. The default frame size is equal to BUFSIZ.

hd (bool) The host uses half-duplex communication; local echo should be performed.

ie (str) Input end-of-file marks. The default is NULL.

md (bool) A hardwired device being used accepts modem control signals. Used when `du` is not present but modem signals are to be used.

oe (str) Output end-of-file string. The default is NULL. When *tip* is transferring a file, this string is sent at end-of-file.

pa (str) The type of parity to use when sending data to the host. The type can be one of even, odd, none, zero (always set bit 8 to zero), or 1 (always set bit 8 to 1). The default is even parity.

pn (str) Telephone numbers for this host. If the telephone number field contains an @ sign, `tip` searches the file `/etc/phones` file for a list of telephone numbers. For further information, see `phones`(5).

tc (str) Indicates that the list of capabilities is continued in the named description. This is used primarily to share common capability information.

Here is a short example showing the use of the capability continuation feature:

```
UNIX-1200:\
        :dv=/dev/ttyd0:el=^D^U^C^S^Q^O:.br
        :du:at=df112:ie=#$%:oe=^D:br#1200:
ourvax|ox:\
        :pn=7654321:tc=UNIX-1200
```

Files

`/etc/remote`

See Also

tip(1c), acucap(5), phones(5)

Name

resolv.conf – resolver configuration file

Description

The resolver configuration file, `/etc/resolv.conf`, contains information that the resolver routines read the first time they are invoked by a process. The resolver file contains ASCII text and lists the name-value pairs that provide various types of resolver information.

The `/etc/resolv.conf` file is required if your system is running BIND. This file must contain the BIND domain name for the local area network. If your system is a BIND client, this file must also contain nameserver entries.

There are two entry formats for the `/etc/resolv.conf` file:

domain *binddomain*

This line specifies the default domain to append to local host names. If no domain entries are present, the domain returned by `gethostname` after the first dot (.) is used. If the host name does not contain a domain, the root domain is assumed.

nameserver *address*

In this entry, the address is the IP address, in dot notation, of the BIND server that should be queried to resolve host name and address information. You should have at least one name server listed. Two or more name servers reduces the possibility of interrupted BIND service in the event that one of the servers is down. You can list up to `NSMAX` (10) name servers. If more than one server is listed, the resolver library queries you to try them in the order listed. If no name server entries are present, the default is to use the name server on the local machine.

The algorithm used is to try a name server, and, if the query times out, to try the next, until out of name servers or the query is resolved. The last step is to repeat trying all the name servers until a maximum number of retries has been made or the query has been resolved.

The name value pair must appear on a single line, and the keyword `domain` or `nameserver` must start each line.

Examples

The following is an example of a `/etc/resolv.conf` file:

```
;
; Data file for a client
;
domain        cities.us
nameserver    128.11.22.33
```

Lines beginning with a semicolon (;) are comment lines.

Files

```
/etc/resolv.conf
```

See Also

gethostname(2), resolver(3), named(8)
Guide to the BIND Service

Name

rhosts – list of hosts that are logically equivalent to the local host

Syntax

/$HOME/.rhosts

Description

The `.rhosts` file allows a user who has an account on the local host to log in from a remote host without supplying a password. It also allows remote copies to the local host.

If the `.rhosts` file exists, it is located in a user's home directory. It is not a mandatory file, however.

The format of a `.rhosts` file entry is:

hostname [*username*]

The *hostname* is the name of the remote host from which the user wants to log into the local host. The *username* is the user's login name on the remote host. If you do not specify a user name, the user must have the same login name on both the remote and local hosts.

The host names listed in the `.rhosts` file may optionally contain the local BIND domain name. For more information on BIND, see the *Guide to the BIND/Hesiod Service*.

If a user `ginger` is logged in to `host1`, and wants to log in to a host called `machine1` without supplying a password, she must:

- Have an account on `machine1`
- Create a `.rhosts` file in her home directory on `machine1`
- Specify **`host1 ginger`** as an entry in the `.rhosts` file.

  If `ginger` has the same login on both `host1` and `machine1`, she can simply specify **`host1`** in her `.rhosts` entry.

NOTE

You can allow the superuser of a remote system to log in to your system without password protection or perform a remote copy by having a `.rhosts` file in the root (/) directory, but it is not recommended.

In addition to having a `.rhosts` file, the superuser needs a `secure` terminal entry in the `/etc/ttys` file for each pseudoterminal configured in the system. The secure entry looks similar to the following:

```
ttyp3       none     network          secure
```

See the `ttys`(5) reference page for more information.

Examples

The following is a sample `.rhosts` file for the user `ginger`. It is located in her home directory on `host1`. She also has accounts on the hosts called `machine1`, `system1`, and `host3`. Her login name on `machine1` and `host3` is the same as on `host1`, but her login on `system1` is `gordon`.

To enable `ginger` to log in to `host1` from `machine1`, `system1`, and `host3` without supplying a password, her `.rhosts` on `host1` should contain the following entries:

```
machine1
system1  gordon
host3
```

See Also

hosts.equiv(5), ttys(5)
Introduction to Networking and Distributed System Services

Name

rmtab – table of local file systems mounted by remote NFS clients

Description

The `rmtab` file resides in the `/etc` directory and contains a list of all remote hosts that have mounted local file systems using the NFS protocols. Whenever a client performs a remote mount, the server machine's mount daemon makes an entry in the server machine's `rmtab` file. The `umount` command removes remotely mounted file system entries. The `umount–a` command broadcasts to all servers and informs them that they should remove all entries from `rmtab` created by the sender of the broadcast message. By placing a `umount–a` command in `/etc/rc.local`, `rmtab` tables on NFS servers can be purged of entries made by a crashed client, who, upon rebooting, did not remount the same file systems that it had before the system crashed. The `rmtab` table is a series of lines of the form:

```
hostname:directory
```

This table is used only to preserve information between crashes and is read only by `mountd`(8nfs) when it starts up. The `mountd` daemon keeps an in-core table, which it uses to handle requests from programs like `showmount`(8nfs) and `shutdown`(8).

Restrictions

Although the `rmtab` table is close to the truth, it may contain erroneous information if NFS client machines fail to execute `umount–a` when they reboot.

Files

`/etc/rmtab`

See Also

mount(8nfs), mountd(8nfs), showmount(8nfs), shutdown(8)

rpc(5)

Name

rpc – remote procedure call file

Description

The `rpc` file is an ASCII file that contains the following information:

rpc name
numerical rpc ID
aliases

Each `rpc` name is separated from the next by a new line. Items are separated by any number of blanks or tab characters or both. A number sign (#) indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines that search the file.

The `rpc` database can be distributed in a network by a naming service, such as Yellow Pages or BIND/Hesiod. See the *Guide to the Yellow Pages Service* or the *Guide to the BIND/Hesiod Service* for setup information.

Files

`/etc/rpc`

See Also

getrpcent(3n)
Guide to the BIND/Hesiod Service
Guide to the Yellow Pages Service

Name

sccsfile – format of SCCS file

Description

An SCCS file is an ASCII file that consists of six logical parts. These six parts include checksum, delta table (contains information about each delta), user names (contains login names and/or numerical group IDs of users who may add deltas), flags (contains definitions of internal keywords), comments (contains arbitrary descriptive information about the file), and body (contains the actual text lines intermixed with control lines).

Throughout an SCCS file there are lines that begin with the **ASCII SOH** (start of heading) character (octal 001). This character is hereafter referred to as the *control character* and will be represented graphically as @. Any line described that is not shown beginning with the control character is prevented from beginning with the control character.

Entries of the form `DDDDD` represent a 5-digit string number between 00000 and 99999.

The logical parts of an SCCS file, described in detail, are:

Checksum

The checksum is the first line of an SCCS file. The form of the line is:

```
@hDDDDD
```

The value of the checksum is the sum of all characters, except those of the first line. The `@h` provides a *magic number* of (octal) 064001.

Delta table

The delta table consists of a variable number of entries of the form:

```
@s DDDDD/DDDDD/DDDDD
@d <type> <SCCS ID> yr/mo/da hr:mi:se <pgmr> DDDDD DDDDD
@i DDDDD ...
@x DDDDD ...
@g DDDDD ...
@m <MR number>
  .
  .
  .
@c <comment> ...
  .
  .
  .
@e
```

The first line (@s) contains the number of lines inserted/deleted/unchanged, respectively. The second line (@d) contains the type of the delta (normal: `D` and removed: `R`), the SCCS ID of the delta, the date and time of creation of the delta, the login name corresponding to the real user ID at the time the delta was created, and the serial numbers of the delta and its predecessor, respectively.

The `@i`, `@x`, and `@g` lines contain the serial numbers of deltas included, excluded, and ignored, respectively. These lines are optional. The `@m` lines (optional) each

contain one MR number associated with the delta. The @c lines contain comments associated with the delta.

The @e line ends the delta table entry.

User names
The list of login names and numerical group ID of users who may add deltas to the file, separated by newlines. The lines containing these login names and numerical group ID are surrounded by the bracketing lines @u and @U. An empty list allows anyone to make a delta.

Flags Keywords used internally. See admin(1) for more information on their use. Each flag line takes the form:

@f <flag> <optional text>

The following flags are defined:

```
@f t   <type of program>
@f v   <program name>
@f i
@f b
@f m   <module name>
@f f   <floor>
@f c   <ceiling>
@f d   <default-sid>
@f n
@f j
@f l   <lock-releases>
@f q   <user defined>
@f z   <reserved for use in interfaces>
```

The t flag defines the replacement for the %Y% identification keyword.

The v flag controls prompting for MR numbers, in addition to comments. If the optional text is present, it defines an MR number-validity checking program.

The i flag controls the warning/error aspect of the ''No id keywords'' message. When the i flag is not present, this message is only a warning; when the i flag is present, this message will cause a fatal error (the file will not be retrieved or the delta will not be made).

When the b flag is present, the -b option can be specified with the get command to cause a branch in the delta tree.

The m flag defines the first choice for the replacement text of the %M% identification keyword.

The f flag defines the ''floor'' release: the release below which no deltas may be added.

The c flag defines the ''ceiling'' release: the release above which no deltas may be added.

The d flag defines the default SID to be used when none is specified on a get command.

The n flag causes *delta* to insert a null delta (a delta that applies *no* changes) in those releases that are skipped when a delta is made in a *new* release. For example, when delta 5.1 is made after delta 2.7, releases 3 and 4 are skipped. The absence of the n

flag causes skipped releases to be completely empty.

The `j` flag causes `get` to allow concurrent edits of the same base I.

The `l` flag defines a list of releases that are locked against editing `get` with the `-e` option.

The `q` flag defines the replacement for the `%Q%` identification keyword.

The `z` flag is used in certain specialized interface programs.

Comments

Arbitrary text surrounded by the bracketing lines `@t` and `@T`. The comments section typically contains a description of the file's purpose.

Body

The body consists of text lines and control lines. Text lines do not begin with the control character; control lines do. There are three kinds of control lines: *insert*, *delete*, and *end*, represented by the following:

```
@I DDDDD
@D DDDDD
@E DDDDD
```

The digit string is the serial number corresponding to the delta for the control line.

See Also

(1), delta(1), get(1), prs(1), sccs(1)
An Introduction to the Source Code
Eric Allman, Supplementary Documentation, Vol. II.

Name

scnhdr – section header for a MIPS object file

Syntax

#include < scnhdr.h>

Description

Every MIPS object file has a table of section headers that specify the layout of the data in the file. Each section that is in an object file has its own header. The C structure appears as follows:

```
struct scnhdr
{
char            s_name[8];    /* section name */
long            s_paddr;      /* physical address, aliased s_nlib */
long            s_vaddr;      /* virtual address */
long            s_size;       /* section size */
long            s_scnptr;     /* file ptr to raw data for section */
long            s_relptr;     /* file ptr to relocation */
long            s_lnnoptr;    /* file ptr to gp table */
unsigned short  s_nreloc;     /* number of relocation entries */
unsigned short  s_nlnno;      /* number of gp table entries */
long            s_flags;      /* flags */
};
```

File pointers are byte offsets into the file; they can be used as the offset in a call to FSEEK (see `ldfcn`(5)). If a section is initialized, the file contains the actual bytes. An uninitialized section is somewhat different. It has a size, symbols defined in it, and symbols that refer to it. It cannot have relocation entries or data. Consequently, an uninitialized section does not contain data in the object file, and the values for *s_scnptr*, *s_relptr*, and *s_nreloc* are zero.

The entries that refer to line numbers (*s_lnnoptr* and *s_nlnno*) are not used for line numbers on MIPS machines. See the header file `sym.h` for the entries to get to the line number table. The entries that were for line numbers in the section header are used for gp tables on MIPS machines.

The number of relocation entries for a section is found in the *s_nreloc* field of the section header. This field is a C language short and can overflow with large objects. If this field overflows, the section header *s_flags* field has the S_NRELOC_OVFL bit set. In this case, the true number of relocation entries is found in the *r_vaddr* field of the first relocation entry for that section. That relocation entry has a type of R_ABS; thus, it is ignored when the relocation takes place.

The gp table gives the section size corresponding to each applicable value of the compiler option `-G` *num* (always including 0), sorted by smallest size first. It is pointed to by the *s_lnnoptr* field in the section header and its number of entries (including the header) is in the *s_nlnno* field in the section header. This table only needs to exist for the *.sdata* and *.sbss* sections. If a small section does not exist,

then the gp table for it is attached to the corresponding large section so the information still gets to the link editor, `ld`. The C union for the gp table follows:

```
union gp_table
{
struct {
        long    current_g_value;    /* actual value */
        long    unused;
} header;
struct {
        long    g_value;            /* hypothetical value */
        long    bytes;              /* section size corresponding
                                       to hypothetical value */
} entry;
};
```

Each gp table has one header structure that contains the actual value of the `-G` *num* option used to produce the object file. An entry must exist for every `-G` *num* **-G** *num* option. The applicable values are all the sizes of the data items in that section.

For *.lib* sections, the number of shared libraries is in the *s_nlib* field (an alias to *s_paddr*). The *.lib* section is made up of *s_nlib* descriptions of shared libraries. Each description of a shared library is a *libscn* structure followed by the path name to the shared library. The C structure appears here and is defined in *scnhdr.h :*

```
struct libscn
{
long    size;         /* size of this entry (including target name) */
long    offset;       /* offset from start of entry to target name */
long    tsize;        /* text size in bytes, padded to DW boundary */
long    dsize;        /* initialized data size */
long    bsize;        /* uninitialized data */
long    text_start;   /* base of text used for this library */
long    data_start;   /* base of data used for this library */
long    bss_start;    /* base of bss used for this library */
/* pathname of target shared library */
};
```

See Also

ld(1), fseek(3s), a.out(5), reloc(5)

services (5)

Name

services – service name file

Description

The `services` file is an ASCII file that contains information regarding the known services available in the DARPA Internet. For each service, a single line should be present with the following information:

official service name
port number
protocol name
aliases

Each `service` name is separated from the next by a new line. Items are separated by any number of blanks or tab characters or both. The port number and protocol name are considered a single item; a backslash (\) is used to separate the port and protocol (for example, 512/tcp). A number sign (#) indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines that search the file.

Service names can contain any printable character other than a field delimiter, newline, or comment character.

The `services` database may be distributed in a network by a naming service, such as Yellow Pages or BIND/Hesiod. See the *Guide to the Yellow Pages Service* or the chapter on Hesiod in the *Guide to the BIND/Hesiod Service* for setup information.

Files

`/etc/services`

See Also

getservent(3n)
Guide to the BIND/Hesiod Service
Guide to the Yellow Pages Service

Name

snmpd.conf – Simple Network Management Protocol (SNMP) daemon configuration file

Description

The `/etc/snmpd.conf` file is a configuration file that contains information used by the `snmpd` daemon to define the static variables whose values are not available in the kernel. It is created for you when you run the `snmpsetup` command, or you can create it manually. The system or network manager is usually responsible for maintaining it.

The following are the `/etc/snmpd.conf` file variables and their significance:

sysDescr *ID-string*

The `sysDescr` variable describes the host. The *ID-string* is the value of the variable `mgmt.mib.system.sysDescr`. The default `sysDescr` entry is of the form:

hostname:machine-type:software id

For example, the `sysDescr` entry for a MicroVAX II named `host1` that is running ULTRIX Version 4.0 might read:

```
sysDescr      host1:MicroVAXII:ULTRIX V4.0 (Rev 64) System #2
```

interface speed *name speed*

The `interface speed` variable describes a value for the Management Information Base (MIB) variable defined as:

`mgmt.mib.interfaces.ifTable.ifEntry.ifSpeed`

The *name* parameter must be an ULTRIX interface name, such as sl0. The following interfaces have default interface speeds and types: de, qe, ni, ln, se, scs, xna. If your system has one of these interfaces, you do not need to specify this parameter.

The *speed* parameter is a decimal number that describes the speed of the link in bits per second. If you do not specify the *speed* parameter, `snmpd` does not return this variable and marks it as unavailable.

interface type *name type*

The `interface type` variable describes a value for the MIB variable in the interface table defined as:

`mgmt.mib.interfaces.ifTable.ifEntry.ifType`

The *name* parameter must be an ULTRIX interface name, such as sl0.

See the sample `/etc/snmpd.conf` file in the Examples section for a complete listing of the possible *type* specifications. If the *type* parameter is not specified, `snmpd` marks it as unavailable.

tcpRtoAlgorithm *algorithm-type*

The `tcpRtoAlgorithm` variable describes a value for the MIB variable in the tcp group defined as `mgmt.mib.tcp.tcpRtoAlgorithm`. This variable defines the Retransmission Time-Out (RTO) algorithm your system uses.

The *algorithm-type* parameter is a numeric code that represents the type of RTO algorithm you are using. The default algorithm is Van Jacobson's, which is algorithm number 4. See the sample `snmpd.conf` file in the Examples section for a listing of the other algorithms.

If you do not specify this parameter, `snmpd` does not return this variable, and marks it as unavailable.

community *name IP-address type*

The `community` variable describes an SNMP community for the agent.

The *name* parameter is a string that describes the name of the community.

The *IP-address* parameter is the dot-notation Internet Protocol (IP) address for the server. Only SNMP packets coming from that address are accepted. If you specify 0.0.0.0 in the *address* field, the SNMP agent honors the request from any Network Management Station (NMS) having the *name* community.

The *type* parameter can be one of the following:

read-only
Permits only monitoring of variables.

read-write
Permits both monitoring and setting of variables.

traps
Generates traps when appropriate and sends them to the specified address with the specified community name.

The possible traps currently generated are cold start and authentication failure.

If you do not specify any community, `snmpd` uses the default community `public` with an address 0.0.0.0 and a read-only type. Invalid uses of communities are logged with the `syslog` command. To limit the use of a community to a finite group of machines, specify another community clause with the same community parameter *name* and a different address.

timeout *value*

The `timeout` variable indicates the timeout value in seconds between the Agent and the Extended Agent. If the Agent does not receive a response within the allotted time, it returns an error message to the NMS. The default timeout value is 5 seconds.

extension *extended-agent p1 p2 p3 p4*

The `extension` variable lists the pathname of the *extended-agent* that the Agent activates.

Variables *p1* through *p4* are passed by the Agent to the Extended Agent; *p1* is usually the process name to be given to the Extended Agent.

Examples

The default `snmpd.conf` file contains only the following entry:

```
community public 0.0.0.0    read-only
```

The following is an example of an extensive `snmpd.conf` file:

```
#
#    snmpd.conf file
#
sysDescr        host1:MicroVAXII:ULTRIX V4.0 System #2


#
#  Describe the TCP RTO algorithm you are using.  Values
#  are listed in RFC 1066, under the TCP group variable:
#  tcpRtoAlgorithm
#  They are:
#
#  other    (1)  -- None of the below
#  constant (2)  -- constant RTO
#  rsre     (3)  -- MILSTD 1778, appendix B
#  vanj     (4)  -- Van Jacobson's algorithm
#
tcprtoalgorithm     4

#
#  Describe who can use your SNMP daemon by
#  defining "communities".    USAGE:
#
#  community <name><IP address><type>
#
#  This is a limited-use community; a finite number of
#  hosts can use it.
#  Can only query from this community.
#
community test1     128.45.10.100 read-only
community test1     128.45.10.101 read-only

#
#  These are our wide-open, general-use communities.  Specifying
#  0.0.0.0 means that any address can use this community only
#  to monitor variables.
#
community public 0.0.0.0 read-only

#
#  This is our only management community.  You can set variables
#  as well as monitor variables with this community.  It is a
#  wide-open community as well.
#
community testwrite 0.0.0.0 read-write

#
#  This is a trap community.  We send traps to these addresses
#  all from the same community name.  Note that a 0.0.0.0
#  address in a trap session is illegal and snmpd will ignore
#  that community definition.
#
community trap1     128.45.10.100 traps
community trap1     128.45.10.101 traps

#
#  The interface speed is given in bits/sec.  USAGE:
#
#  interface speed <name> <speed>
#
interface speed sl0 9600
```

```
#
#  The <name> parameter for the interface type is the
#  same as the <name> for the interface speed, sl0 for
#  this example.  USAGE:
#
#  interface type <name> <type>
#
#  The code number for the proper interface hardware type
#  is specified in RFC 1066 under the ifType object
#  definition.
#
#  Some possible values:
#
#  other                    (1)
#  regular1822              (2)
#  hdh1822                  (3)
#  ddn-x25                  (4)
#  rfc877-x25               (5)
#  ethernet-csmacd          (6)
#  iso88023-csmacd          (7)
#  iso88024-tokenBus        (8)
#  iso88025-tokenRing       (9)
#  iso88026-man             (10)
#  starLan                  (11)
#  Proteon-10MBit           (12)
#  Proteon-80MBit           (13)
#  hyperchannel             (14)
#  fddi                     (15)
#  lapb                     (16)
#  sdlc                     (17)
#  t1-carrier               (18)
#  cept                     (19) -- European equivalent of T-1
#  basicIsdn                (20)
#  primaryIsdn              (21)
#  propPointToPointSerial   (22) -- proprietary serial
#
interface type  sl0 1

#
# Timer value to time out requests to extended agents.
#
timeout      6

#
# List of extended agents.
#
extension /etc/snmpextd          snmpextd
```

In this example, note the following:

- Community `test1` can be monitored by either 128.45.10.100 or 128.45.10.101.
- Community `public` can be monitored by any NMS.
- Community `testwrite` can be monitored and managed by any NMS.
- When a trap is generated, it is sent to community `trap1` at 128.45.10.100 or 128.45.10.101. Destination addresses must have a mechanism in place to handle the traps.

See Also

snmpext(3n), snmpd(8n), snmpsetup(8n)
RFC 1066—*Management Information Base for Network Management of TCP/IP-based Internets*
RFC 1067—*A Simple Network Management Protocol*
Guide to Networking

Name

stab – symbol table types

Syntax

#include <stab.h>

Description

The `stab.h` file defines some values of the n_type field of the symbol table of `a.out` files. These are the types for permanent symbols (that is, not local labels, and so on) used by the debugger `dbx` and the Berkeley Pascal compiler `pc`(1). Symbol table entries can be produced by the `.stabs` assembler directive, which allows you to specify a double-quote delimited name, a symbol type, one char and one short of information about the symbol, and an unsigned long (usually an address).

To avoid having to produce an explicit label for the address field, the .stabd directive can be used to implicitly address the current location. If no name is needed, symbol table entries can be generated using the .stabn directive. The loader promises to preserve the order of symbol table entries produced by .stab directives. As described in `a.out`(5), an element of the symbol table consists of the following structure:

```
/*
struct nlist {
   union {
        char   *n_name; /* for use when in-core */
        long    n_strx; /* index into file string table */
   } n_un;
        unsigned char   n_type;    /* type flag */
        char            n_other;   /* unused */
        short           n_desc;    /* see struct desc, below */
        unsigned        n_value;   /* address or offset or line */
};
```

The low bits of the n_type field are used to place a symbol into one segment, maximum, according to the following masks defined in <a.out.h>. If none of the segment bits are set, a symbol cannot be in any of these segments.

```
* Simple values for n_type.
#define N_UNDF  0x0     /* undefined */
#define N_ABS   0x2     /* absolute */
#define N_TEXT  0x4     /* text */
#define N_DATA  0x6     /* data */
#define N_BSS   0x8     /* bss */

#define N_EXT   01      /* external bit, or'ed in */
```

The n_value field of a symbol is relocated by the linker, `ld`, as an address within the appropriate segment. N_value fields of symbols not in any segment are unchanged by the linker. In addition, the linker will discard certain symbols, according to rules of its own, unless the n_type field has one of the following bits set:

```
/*
* Other permanent symbol table entries have some of the
* N_STAB bits set.  These are given in <stab.h>
*/
#define N_STAB 0xe0 /* if any of these bits set, don't discard */
```

This allows up to 112 (7 * 16) symbol types, split between the various segments. Some of these have already been claimed. The C compiler generates the following n_type values, where the comments give the use for *.stab*s and the n_name, n_other, n_desc, and n_value fields of the given n_type:

```
#define N_GSYM  0x20 /* global symbol: name,,0,type,0 */
#define N_FNAME 0x22 /* procedure name (f77 kludge): name,,0 */
#define N_FUN   0x24 /* procedure: name,,0,linenumber,address */
#define N_STSYM 0x26 /* static symbol: name,,0,type,address */
#define N_LCSYM 0x28 /* .lcomm symbol: name,,0,type,address */
#define N_RSYM  0x40 /* register sym: name,,0,type,register */
#define N_SLINE 0x44 /* src line: 0,,0,linenumber,address */
#define N_SSYM  0x60 /* struct elt: name,,0,type,struct_offset */
#define N_SO    0x64 /* source file name: name,,0,0,address */
#define N_LSYM  0x80 /* local sym: name,,0,type,offset */
#define N_SOL   0x84 /* #included file name: name,,0,0,address */
#define N_PSYM  0xa0 /* parameter: name,,0,type,offset */
#define N_ENTRY 0xa4 /* alt entry: name,linenumber,address */
#define N_LBRAC 0xc0 /* lft bracket: 0,,0,nesting level,address */
#define N_RBRAC 0xe0 /* rt bracket: 0,,0,nesting level,address */
#define N_BCOMM 0xe2 /* begin common: name,, */
#define N_ECOMM 0xe4 /* end common: name,, */
#define N_ECOML 0xe8 /* end common (local name): ,,address */
#define N_LENG  0xfe /* second stab entry with length information */
```

The n_desc holds a type specifier in the form used by the Portable C Compiler, cc(1), in which a base type is qualified in the following structure:

```
struct desc {
       short  q6:2,
              q5:2,
              q4:2,
              q3:2,
              q2:2,
              q1:2,
              basic:4;
};
```

There are 4 qualifications, with q1 the most significant and q6 the least significant:

0 None
1 Pointer
2 Function
3 Array

The 16 basic types are assigned as follows:

0 Undefined
1 Function argument
2 Character
3 Short
4 Int
5 Long
6 Float
7 Double
8 Structure
9 Union
10 Enumeration
11 Member of enumeration

12 Unsigned character
13 Unsigned short
14 Unsigned int
15 Unsigned long

The same information is encoded in a more useful form in the symbolic string. The symbol's name is followed by a colon, which is followed by a description of the symbol's type. This begins with one of the following letters:

c Constant
f Local function
F Function name
G Global variable
p Argument (by value)
P External procedure
r Register variable
s Static variable
t Typedef name
T Local variable
v Argument (by ref)
V Local static variable
No letter
Local dynamic variable

This is followed by the variable's type, where *type* is any of the following:

integer Same as previously defined type *integer*

integer=type Define type *integer* to have form *type*

type Pointer to *type*

r*type;low;high;* Range of *type* from *low* to *high*

a*rangetype* Array with bounds *range* of *type*

e*name:value,;* Enumerated type. The phrase "*name:value,*" repeats as needed.

s*sizename:type,offset,size;;*
Structure. The *size* is the number of bytes in the complete structure. The phrase "*name:type,offset,size;*" repeats as needed, giving the *offset* from the start of the structure (in bits) and the *size* in bits of each member.

u*sizename:type,offset,size;;*
Union. Analagous to structure entry.

S*type* Set of *type*.

f*type,integer;type,class;*
Function returning *type* with *integer* parameters, described by the repeating "*type,class;*" phrase.

p*integer;type,class;*
Procedure–like function

d*type* File of *type*

The Berkeley Pascal compiler, pc(1), uses the following n_type value:

```
#define      N_PC   0x30   /* global pascal symbol: name,,0,subtype,line */
```

The complier uses the following subtypes to do type checking across separately compiled files:

1 Source file name
2 Included file name
3 Global label
4 Global constant
5 Global type
6 Global variable
7 Global function
8 Global procedure
9 External function
10 External procedure
11 Library variable
12 Library routine

See Also

as(1), cc(1), dbx(1), ld(1), pc(1), a.out(5)

statmon (5)

Name

statmon, current, backup, state – statd directory and file structures

Syntax

/etc/sm /etc/sm.bak /etc/state

Description

The directories `/etc/sm` and `/etc/sm.bak` are generated by the `statd` daemon. Each entry in `/etc/sm` represents the name of the system to be monitored by `statd`. Each entry in `/etc/sm.bak` represents the name of the system to be notified by `statd` upon its recovery.

The file `/etc/state` is generated by `statd` to record its version number, that is, the number of times `statd` was invoked. The version number is incremented each time a crash or recovery takes place.

See Also

statd(8c), lockd(8c)

Name

stl_comp – software subset compression file

Description

The software subset compression file is used to indicate to the `setld` utility that the subsets on a distribution are in compressed format. This file is created in the *instctrl* directory of the kit output hierarchy by the `kits` utility if the value for the *COMPRESS* attribute in the key file for the product is 1. The name of this file is of the form *XXXZZZ.comp* where *XXX* is the value of the *CODE* attribute for the product and *ZZZ* is the value of the *VERS* attribute.

See Also

kits(1), stl_key(5), setld(8)

Guide to Preparing Software for Distribution on ULTRIX Systems

stl_ctrl (5)

Name

stl_ctrl – software subset control files

Description

Each software subset that is distributed on media used with the `setld` command has an associated *control file*. This control file is created by the `kits` utility. It contains the attribute information for the subset. Once the subset has been installed on a system, the control file is placed in the `usr/etc/subsets` directory. The control file for each subset has a name formed by appending the extension `.ctrl` to the end of the subset name.

The attribute definitions in the file are stored as attribute name and value pairs separated by an equal sign (=). The attributes are:

NAME
: The name of the product of which this subset is a member. This attribute has the same value as the *NAME* attribute defined for the product in the product attributes section of the key file.

DESC
: The text description of the subset as given in the subset descriptor for this subset in the key file.

NVOLS
: Obsolete.

MTLOC
: A pair of integers separated by a colon (:). These integers are used by the `setld` utility to find the subset on tape media. The first number is the volume in the product tape set; the second is the location within the volume. The values established for this attribute by the `kits` utility are place holders. The values are later updated as the subset is being written to tape by the `gentapes` command.

DEPS
: The dependency list for this subset. The information for this list is taken from the subset descriptor in the key file.

FLAGS
: The flags value from the key file.

Examples

Here are the contents of `UDTBASE400.ctrl`, the control file for the `UDTBASE400` subset:

```
NAME='ULTRIX/UWS T4.0 (RISC)'
DESC='Base System'
NVOLS=1:112
MTLOC=1:1
DEPS="."
FLAGS=1
```

See Also

genra(1), gentapes(1), kits(1), stl_key(5), stl_tape(5), setld(8)
Guide to Preparing Software for Distribution on ULTRIX Systems

stl_key (5)

Name

stl_key – setld kit manufacturing key file

Description

The manufacturing key files are used by the software kitting program `kits` in producing software distribution packages in `setld` format.

A key file has a global data section and a subset descriptor section. The sections are separated by a line that is empty except for the %% character sequence.

The global section contains the product level attributes of the product. Comments are permitted in this section. They begin with the number sign (#) character and end at the next newline character. Attributes are specified by giving the attribute key, an equal sign (=), and a value for the attribute. There must be no white space surrounding the equals sign (=). There are five attributes that must be present with non-null values. These are *NAME, CODE, VERS, MI,* and *ROOT.* An explanation of each of the attributes follows:

NAME
: The name of the product. The value for *NAME* is a string of up to 40 characters.

CODE
: The 3-character product code for the product, for example, ULT.

VERS
: The 3-digit version code for the product, for example, 040.

MI
: The pathname of the master inventory file for the product.

ROOT
: A flag with values of 0 or 1. It is used to determine if a ROOT image checksum should be computed for the product image file. There is no ROOT image in any product other than ULTRIX. Set it to 0.

RXMAKE
: A flag with values of 0 or 1. Setting it to 0 suppresses the manufacture of subset images for distribution on RX50 diskettes. Omitting this attribute from the key file will cause the `kits` program to assume a default value of 1. Digital recommends setting it to 0.

COMPRESS
: A flag with values of 0 or 1. Setting it to 1 causes the subset images to be compressed using the `compress` utility, thereby saving space on the distribution media. Setting it to 0 suppresses compression. Omitting this attribute from the key file will cause the `kits` program to assign a default value of 0.

The subset descriptor section contains one subset descriptor for each subset in the product. There is one subset descriptor per line in this section and comments are not permitted.

A subset descriptor contains subset-specific attributes in four fields separated by TAB (CTRL/I) characters. A description of each field follows:

SUBSET
: This field contains the name of the subset being described by the descriptor. The subset name is composed of the product code, name and version code.

DEPENDENCIES
: A list of subsets on which the described subset depends. If there are no such subsets, the period character (.) is used. Multiple subset

dependencies are separated by a vertical bar character (|).

FLAGS A subset flags value. This is an integer. Bit 0 is used to mark the subset as irremovable. If bit 0 is set, `setld` can never delete the subset. Bit 1 is used to mark the subset as optional, otherwise it is mandatory and must be installed from the media when encountered by `setld`.

DESCRIPTION
This is a description of the subset in 40 or fewer characters. It is used in the menu that `setld` presents to a user installing the software. If spaces are desired in this field, the field must be enclosed in single quotes.

The subset descriptors must be listed in the order in which the subsets are installed by `setld`.

Restrictions

The required attributes and default values are not optimal.

Comments in the subset descriptor section will cause serious problems when encountered by the `kits` program.

TAB formatting in the subset descriptors is tightly enforced by the `kits` program.

Examples

This is an example key file:

```
#  ULW400.k -
#         ULTRIX WS V4.0 Server 2/2 (VAX) Mfg Key File
#
#  "@(#)ULW400.k 2.2 (ULTRIX) 4/12/89"
#
#  000    02-mar-1989   ccb
#  Copy from V2.0 (VAX) Sources
#  Revision update for V4.0

#% PRODUCT-LEVEL ATTRIBUTES
NAME='ULTRIX Worksystem Software V4.0'
CODE=UWS
VERS=400
MI=ULT400.mi
ROOT=0
RXMAKE=0
COMPRESS=1

#% SUBSET-LEVEL ATTRIBUTES
%%
UWSX11400 ULTINET400    0      'X11/DECwindows User Environment'
UWSFONT400       UWSX11400     0      'X11/DECwindows 75dpi Fonts'
UWSFONT15400     .      2      'X11/DECwindows 100dpi Fonts'
UWSDECW400       .      2      'Optional DECwindows Applications'
UWSXDEV400       ULTPGMR400    2      'Worksystems Development Software'
UWSMAN400 ULTDCMT400    2      'UWS Manual Pages'
```

See Also

kits(1), stl_comp(5), stl_ctrl(5), stl_image(5), setld(8)
Guide to Preparing Software for Distribution on ULTRIX Systems

Name

stl_mi – software distribution master inventory file format

Description

The master inventory (mi) files are used by the subset kitting program `kits` when manufacturing subsets for installation with the `setld` utility.

A master inventory file contains one record for each file in a product containing vendor-specified attribute information about each file in the kit.

The master inventory contains ASCII data, one record per line. Each record is composed of three fields, which must be separated by TAB characters.

Here is a description of each of the fields:

FLAGS
: This is an integer flags value. The two lowest bits of this flag are defined. All other bits are reserved for use by Digital.

  Bit 0 of this flag is the precedence bit. Setting the precedence bit for a file indicates to `setld` that a new copy of this file is not as important to the target system as a copy that is already there. This is used by `setld` when determining which files to restore to the system after updating a subset. Files containing configuration information that can be modified after being installed are often marked with this flag.

  Bit 1 of this flag is defined as the volatility bit. It is used to indicate that the file will change after being installed to the target system and that the changes do not indicate that the contents of the file have been corrupted. When an application requires a log file, it is often installed as a zero length file on the target system. Such log files are normally marked in the master inventory with bit 1.

PATHNAME
: This is the name of the file for which the record exists. The pathname in this field must begin with a leading period (.). All records in the file must be sorted in ascending order on this field. There cannot be two records in a master inventory that represent the same file. The `newinv` program enforces these requirements.

SUBSET
: The name of the subset to which the file belongs. A file can be distributed as part of one subset only. Files that share a gnode (links) must be in the same subset. Files that are in the product hierarchy but are not distributed as part of any subset should have a dash (-) in this field.

Examples

This example shows a section of the master inventory used to manufacture a release of the ULTRIX software:

```
0       ./etc/newfs   UDTBASE040
0       ./etc/zoneinfo/Poland       -
0       ./usr/bin/passwd     UDTBASE040
0       ./usr/diskless/dev/rrz2c   UDTDL040
0       ./usr/etc/lockpw     UDTBASE040
0       ./usr/include/search.h      UDTBASE040
0       ./usr/lib/cmplrs/cc2.0/ppu UDTBASE040
```

```
0       ./usr/lib/libplotdumb.a     UDTPGMR040
0       ./usr/lib/terminfo/2/2621-nl      UDTBASE040
0       ./usr/lib/terminfo/a/altoh19      UDTBASE040
0       ./usr/lib/terminfo/h/h19b  UDTBASE040
0       ./usr/lib/terminfo/t/tek4024      UDTBASE040
0       ./usr/man/man1/capsar.1     UDTMAN040
0       ./usr/man/man1/ptx.1        UDTMAN040
0       ./usr/man/man2/listen.2     UDTMAN040
0       ./usr/man/man3/endhostent.3n      UDTMAN040
0       ./usr/man/man3/ldaopen.3x  UDTMAN040
0       ./usr/man/man3/ruserok.3x  UDTMAN040
0       ./usr/man/man3/tparm.3cur  UDTMAN040
0       ./usr/man/man5/tzfile.5     UDTMAN040
0       ./usr/man/man8/secsetup.8  -
0       ./usr/sys/MIPS/BINARY/mc146818clock.o    UDTBIN040
0       ./usr/sys/data/dhu_data.c  UDTBIN040
0       ./usr/sys/h/devio.h UDTBASE040
0       ./usr/sys/io/uba/qduser.h  UDTBIN040
0       ./usr/sys/net/rpc/clnt.h   UDTBIN040
0       ./usr/var/dss/ncs/llbd      UDTRPCRT040
```

See Also

invcutter(1), kits(1), stl_inv(5), iff(8), setld(8).
Guide to Preparing Software for Distribution on ULTRIX Systems

Name

svc.conf - database service selection and security configuration file

Description

The `svc.conf` file is a mandatory system file that allows you to select the desired services on a per database basis. It also allows you to select security parameters. The default `/etc/svc.conf` file has `local` as the service selected for each database. This file must be modified when adding or removing a naming service, such as Yellow Pages or BIND/Hesiod. The valid services are `local`, `yp`, and `bind`. Modifications to the `/etc/svc.conf` file can be made with an editor or the `/usr/etc/svcsetup` command for database service selection. They can be made with the `/usr/etc/secsetup` command for security parameter selection. Changes take effect immediately.

NOTE

The recommended configuration is that you have `local` as the first entry for all databases.

Restrictions

White space is allowed only after commas or newlines.

You must have `local` as the first entry for the `passwd` and `hosts` databases.

You must have `yp` as the entry for the `netgroup` database.

You must have either `local` or `bind` as the entry for the `auth` database.

Examples

The following is a sample `/etc/svc.conf` file:

```
aliases=yp
auth=local,bind
group=local,yp
hosts=local,bind,yp
netgroup=yp
networks=bind
passwd=local,bind
protocols=local,bind
rpc=local,bind
services=local

PASSLENMIN=6
PASSLENMAX=16
SOFTEXP=604800          # 7 days in seconds
SECLEVEL=BSD            # (BSD | UPGRADE | ENHANCED)
```

Files

```
/etc/svc.conf
/usr/sys/h/svcinfo.h
```

See Also

getsvc(3), svcsetup(8)
Guide to the BIND/Hesiod Service
Guide to the Yellow Pages Service

Name

syms – MIPS symbol table

Syntax

#include < sym.h>
#include < symconst.h>

Description

Unlike the COFF symbol table, the MIPS symbol table consists of many tables unbundling information. The symbol table should be viewed as a network-style database designed for space and access efficiency.

The following structures or tables appear in the MIPS symbol table:

TABLE	CONTENTS
Symbolic header	Sizes and locations of all other tables
File descriptors	Per file locations for other tables
Procedure descriptors	Frame information and location of procedure info
Local symbols	Local type, local variable, and scoping info
Local strings	String space for local symbols
Line numbers	Compacted by encoding, contains a line per instruction
Relative file descriptors	Indirection for interfile symbol access
Optimization symbols	To be defined
Auxiliary symbols	Variable data type information for each local symbol
External symbols	Loader symbols (global text and data)
External strings	String space for external symbols
Dense numbers	Index pairs (file, symbol) for compiler use

External and local symbols contain the standard concept of a symbol as follows:

```
struct
{
        long    iss;    /* index into string space */
        long    value; /* address, size, etc., depends on sc and st */
        unsigned        st: 6; /* symbol type (e.g. local, param, etc.) */
        unsigned        sc: 5; /* storage class (e.g. text, bss, etc.) */
        unsigned        reserved: 1;
        unsigned        index; /* index to symbol or auxiliary tables */
};
```

See Also

ldfcn(5)

tar(5)

Name

tar, mdtar – tape archive file format

Description

The tape archive command `tar` dumps several files, including special files, into one, in a medium suitable for transportation.

A `tar` tape or file is a series of blocks. Each block is of size TBLOCK. A file on the tape is represented by a header block, which describes the file, followed by zero or more blocks, which give the contents of the file. At the end of the tape are two blocks filled with binary zeros, as an end-of-file indicator.

The blocks are grouped for physical I/O operations. Each group of *n* blocks (where *n* is set by the `b` option on the `tar`(1) command line, and the default is 20 blocks) is written with a single system call; on 9-track tapes, the result of this write is a single tape record. The last group is always written at the full size, so blocks after the two zero blocks contain random data. On reading, the specified or default group size is used for the first read, but if that read returns less than a full tape block, the reduced block size is used for further reads.

The following is an example of a header block:

```
#define TBLOCK  512
#define NAMSIZ  100

union hblock {
        char dummy[TBLOCK];
        struct header {
                char name[NAMSIZ];
                char mode[8];
                char uid[8];
                char gid[8];
                char size[12];
                char mtime[12];
                char chksum[8];
                char linkflag;
                char linkname[NAMSIZ];
                char rdev[6]
        } dbuf;
};
```

The *name* field is a null-terminated string. The other fields are 0-filled octal numbers in ASCII. Each field (of width *w*) contains *w* minus 2 digits, a space, and a null, except *size* and *mtime* , which do not contain the trailing null. The *name* field specifies the name of the file, as specified on the `tar` command line. Files dumped because they were in a directory that was named in the command line have the directory name as prefix and */filename* as suffix. The `mode` field specifies the file mode, with the top bit masked off. The *uid* and *gid* fields specify the user and group numbers that own the file. The *size* field specifies the size of the file in bytes. Links and symbolic links are dumped with this field specified as zero. The *mtime* field specifies the modification time of the file at the time it was dumped. The *chksum* field is a decimal ASCII value, which represents the sum of all the bytes in the header block. When calculating the checksum, the *chksum* field is treated as if it were all blanks. The *linkflag* field is ASCII 0 if the file is normal or a special file and ASCII 1 if it is a hard link, and ASCII 2 if it is a symbolic link. The name to

which it is linked, if any, is in *linkname,* with a trailing null. Unused fields of the header are binary zeros and are included in the checksum. The *rdev* field encodes the ASCII representation of a device special file's major and minor device numbers.

The first time a given i-node number is dumped, it is dumped as a regular file. The second and subsequent times, it is dumped as a link instead. Upon retrieval, if a link entry is retrieved, but not the file it was linked to, an error message is printed and the tape must be manually rescanned to retrieve the linked file.

The encoding of the header is designed to be portable across machines.

Restrictions

Names or link names longer than NAMSIZ produce error reports and cannot be dumped.

See Also

tar(1)

Name

term – terminal driving tables for nroff

Description

The `nroff`(1) command uses driving tables to customize its output for various types of output devices. These driving tables are ASCII files installed in `/usr/lib/term/tab`*name*, where *name* is the name for that terminal type as given in `term`(7) The file `/usr/lib/term/example` describes the format of each field in the driving table.

Files

`/usr/lib/term/example` decribes the format of each field
`/usr/lib/term/tab`*name* driving tables

See Also

term(7)

Name

term – terminal driving tables for nroff

Description

The nroff(1) command uses driving tables to customize its output for various types of output devices. These driving tables are written as C programs, compiled, and installed in /usr/lib/term/tab*name*, where *name* is the name for that terminal type as given in term(7). The structure of the tables is as follows:

```
#define       INCH   240
struct {
        int bset;
        int breset;
        int Hor;
        int Vert;
        int Newline;
        int Char;
        int Em;
        int Halfline;
        int Adj;
        char *twinit;
        char *twrest;
        char *twnl;
        char *hlr;
        char *hlf;
        char *flr;
        char *bdon;
        char *bdoff;
        char *ploton;
        char *plotoff;
        char *up;
        char *down;
        char *right;
        char *left;
        char *codetab[256-32];
        char *zzz;
} t;
```

The meanings of the various fields are:

bset Bits to set in the *c_oflag* field of the termio structure before output. For further information, see tty(4).

breset Bits to reset in the *c_oflag* field of the termio structure before output.

Hor Horizontal resolution in fractions of an inch.

Vert Vertical resolution in fractions of an inch.

Newline Space moved by a newline (linefeed) character in fractions of an inch.

Char Quantum of character sizes, in fractions of an inch (that is, a character is a multiple of Char units wide).

Em Size of an em in fractions of an inch.

Halfline Space moved by a half-linefeed (or half-reverse-linefeed) character in fractions of an inch.

Adj Quantum of white space, in fractions of an inch (that is, white spaces are a multiple of Adj units wide).

Note: if this is less than the size of the space character (in units of Char; see the following fields for how the sizes of characters are defined), `nroff` outputs fractional spaces using plot mode. Also, if the `-e` option to `nroff` is used, `nroff` sets Adj equal to Hor.

twinit Set of characters used to initialize the terminal in a mode suitable for `nroff`.

twrest Set of characters used to restore the terminal to normal mode.

twnl Set of characters used to move down one line.

hlr Set of characters used to move up one-half line.

hlf Set of characters used to move down one-half line.

flr Set of characters used to move up one line.

bdon Set of characters used to turn on hardware boldface mode, if any.

bdoff Set of characters used to turn off hardware boldface mode, if any.

iton Set of characters used to turn on hardware italics mode, if any.

itoff Set of characters used to turn off hardware italics mode, if any.

ploton Set of characters used to turn on hardware plot mode (for Diablo type mechanisms), if any.

plotoff Set of characters used to turn off hardware plot mode (for Diablo type mechanisms), if any.

up Set of characters used to move up one resolution unit (Vert) in plot mode, if any.

down Set of characters used to move down one resolution unit (Vert) in plot mode, if any.

right Set of characters used to move right one resolution unit (Hor) in plot mode, if any.

left Set of characters used to move left one resolution unit (Hor) in plot mode, if any.

codetab Definition of characters needed to print an `nroff` character on the terminal. The first byte is the number of character units (Char) needed to hold the character; that is, "\001" is one unit wide, "\002" is two units wide, and so on. The high-order bit (0200) is on if the character is to be underlined in underline mode (.ul). The rest of the bytes are the characters used to produce the character in question. If the character has the sign (0200) bit on, it is a code to move the terminal in plot mode. It is encoded as:

0100 bit on	Vertical motion
0100 bit off	Horizontal motion
040 bit on	Negative (up or left) motion
040 bit off	Positive (down or right) motion
037 bits	Number of such motions to make

zzz A zero terminator at the end

All quantities that are in units of fractions of an inch should be expressed as INCH**num/denom*, where *num* and *denom* are respectively the numerator and denominator of the fraction; that is, 1/48 of an inch would be written as "INCH1/48".

If any sequence of characters does not pertain to the output device, that sequence should be given as a null string.

The source code for the terminal *name* is in `/usr/src/usr.bin/nroff/term/tab`*name*. When a new terminal type is added, the file `maketerms.c` should be updated to include the source to that driving table (use #include). Note that the various terminal types are grouped into "parts" labelled PART1, PART2, and PART3. If necessary, more parts can be added. Users can make other changes to `maketerms.c` as needed. The makefile `terms.mk` in that directory should then be updated.

Files

`/usr/lib/term/tab`*name*	Driving tables
`tab`*name*`.c`	Source for driving tables

See Also

tty(4), term(7)

termcap(5)

Name

termcap – terminal capability data base

Syntax

/etc/termcap

Description

The `termcap` file is a data base describing terminals used, for example, by `vi`(1) and `curses`(3x). Terminals are described in `termcap` by giving a set of capabilities which they have and by describing how operations are performed. Padding requirements and initialization sequences are included in `termcap`.

Entries in `termcap` consist of a number of fields separated by colons (:). The first entry for each terminal gives the names which are known for the terminal, separated by vertical bars (|). The first name is always 2 characters long and is used by older Version 6 systems, which store the terminal type in a 16-bit word in a system-wide data base. The second name given is the most common abbreviation for the terminal, and the last name given should be a long name fully identifying the terminal. The second name should contain no blanks. The last name may contain blanks for readability.

Capabilities

(P) indicates padding is commonly needed for these strings.
(P*) indicates that padding may be based on the number of lines affected.

Name	Type	Pad?	Description
ae	str	(P)	End alternate character set.
al	str	(P*)	Add new blank line.
am	bool		Terminal has automatic margins.
as	str	(P)	Start alternate character set.
bc	str		Backspace, if not CTRL/H.
bl	str		Audible bell character.
bs	bool		Terminal can backspace with CTRL/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.
ca	bool		Cursor addressable.
cd	str	(P*)	Clear to end of display.
ce	str	(P)	Clear to end of line.
ch	str	(P)	Like `cm`, but horizontal motion only; line stays the same.
cl	str	(P*)	Clear screen.
cm	str	(P)	Cursor motion.
co	num		Number of columns in a line.
cr	str	(P*)	Carriage return (default CTRL/M).
cs	str	(P)	Change scrolling region (VT100); like `cm`.
ct	str		Clear all tab stops.
cv	str	(P)	Like `ch`, but vertical only.
da	bool		Display may be retained above.
dB	num		Number of millisec of `bs` delay needed.

db	bool		Display may be retained below.
dC	num		Number of millisec of `cr` delay needed.
dc	str	(P*)	Delete character.
dF	num		Number of millisec of `ff` delay needed.
dl	str	(P*)	Delete line.
dm	str		Delete mode (enter).
dN	num		Number of millisec of `nl` delay needed.
do	str		Move down one line.
ds	str		Clear host writable status line.
dT	num		Number of millisec of `ta` delay needed.
ed	str		End delete mode.
ei	str		End insert mode; give ":ei=:" if `ic`.
eo	str		Can erase overstrikes with a blank.
es	bool		Standout mode allowed on host writable status line.
ff	str	(P*)	Hard-copy terminal page eject (default CTRL/L).
fs	str		Close host writable status line to writing.
gt	bool		Gtty indicates tabs.
hc	bool		Hard-copy terminal.
hd	str		Half-line down (forward 1/2 linefeed).
ho	str		Home cursor (if no `cm`).
hs	bool		Host writable status line capabilities.
hu	str		Half-line up (reverse 1/2 linefeed).
hz	str		Hazeltine; cannot print tildes (~).
ic	str	(P)	Insert character.
if	str		Name of file containing `is`.
im	bool		Insert mode (enter); give ":im=:" if `ic`.
in	bool		Insert mode distinguishes nulls on display.
ip	str	(P*)	Insert pad after character inserted.
is	str		Terminal initialization string.
k0-k9	str		Sent by "other" function keys 0-9.
kb	str		Sent by backspace key.
kd	str		Sent by terminal down arrow key.
ke	str		Out of "keypad transmit" mode.
kh	str		Sent by home key.
kl	str		Sent by terminal left arrow key.
kn	num		Number of "other" keys.
ko	str		Termcap entries for other nonfunction keys.
kr	str		Sent by terminal right arrow key.
ks	str		Put terminal in "keypad transmit" mode.
ku	str		Sent by terminal up arrow key.
l0-l9	str		Labels on "other" function keys.
le	str		Move cursor left one place.
li	num		Number of lines on screen or page.
ll	str		Last line, first column (if no `cm`).
ma	str		Arrow key map, used by `vi` Version 2 only.
mb	str		Turn on blinking.
md	str		Enter bold (extra-bright) mode.
me	str		Turn off all attributes, normal mode.
mh	str		Enter dim (half-bright) mode.
mi	bool		Safe to move while in insert mode.
ml	str		Memory lock on above cursor.
mr	str		Enter reverse mode.

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		Nondestructive space (cursor right).
nl	str	(P*)	Newline character (default \n).
ns	bool		Terminal is a CRT, but does not scroll.
os	bool		Terminal overstrikes.
pc	str		Pad character (rather than null).
pt	bool		Has hardware tabs (may need to be set with is).
rc	str		Recover from last save cursor (sc).
rf	str		Reset file, like initialization file (if) but for reset.
rs	str		Reset string, like initialization string (is) but for reset.
sc	str		Save cursor.
se	str		End stand out mode.
sf	str	(P)	Scroll forwards.
sg	num		Number of blank chars left by so or se.
so	str		Begin stand out mode.
sr	str	(P)	Scroll reverse (backwards).
st	str		Save cursor.
ta	str	(P)	Tab (other than CTRL/I or with padding).
tc	str		Entry of similar terminal – must be last.
te	str		String to end programs that use cm.
ti	str		String to begin programs that use cm.
ts	str		Open host writable status line to writing.[jA.
uc	str		Underscore one char and move past it.
ue	str		End underscore mode.
ug	num		Number of blank chars left by us or ue.
ul	bool		Terminal underlines even though it does not overstrike.
up	str		Upline (cursor up).
us	str		Start underscore mode.
vb	str		Visible bell (may not move cursor).
ve	str		Sequence to end open/visual mode.
vs	str		Sequence to start open/visual mode.
vt	num		Virtual terminal number.
xb	bool		Beehive (f1=escape, f2=CTRL/C).
xn	bool		A newline is ignored after a wrap (Concept).
xr	bool		Return acts like ce \r \n (Delta Data).
xs	bool		Standout not erased by writing over it (HP 264?).
xt	bool		Tabs are destructive, magic so char (Teleray 1061).

A Sample Entry

The following entry, which describes the Concept–100, is among the more complex entries in the termcap file as of this writing. This particular 'Concept' entry is outdated and is used as an example only:

```
c1 | c100 | concept100:is=\EU\Ef\E7\E5\E8\El\ENH\EK\E\200\Eo&\200:\
      :al=3*\E^R:am:bs:cd=16*\E^C:ce=16\E^S:cl=2*^L:cm=\Ea%+ %+\
      :co#80:\ :dc=16\E^A:dl=3*\E^B:ei=\E\200:eo:im=\E^P:in\
      :ip=16*:li#24:mi:nd=\E=:\ :se=\Ed\Ee:so=\ED\EE:ta=8\t\
      :ul:up=\E;:vb=\Ek\EK:xn:
```

Entries can continue onto multiple lines by giving a backslash (\) as the last character of a line. Empty fields can be included for readability (here between the last field on a line and the first field on the next).

Types of Capabilities

Capabilities in `termcap` are of three types: Boolean capabilities that indicate that the terminal has some particular feature; numeric capabilities giving the size of the terminal or the size of particular delays; and string capabilities, which give a sequence that can be used to perform particular terminal operations.

All capabilities have 2-letter codes. For instance, the fact that the Concept has "automatic margins" (that is, an automatic return and linefeed when the end of a line is reached) is indicated by the capability `am`. Hence, the description of the Concept includes `am`. Numeric capabilities are followed by the number sign (#) and then the value. Thus, `co`, which indicates the number of columns the terminal has, gives the value '80' for the Concept.

Finally, string-valued capabilities, such as `ce` (clear to end-of-line sequence), are given by the 2-character code: an equal sign (=) and then a string ending at the next following colon (:). A delay in milliseconds may appear after the equal sign (=) in such a capability. Padding characters are supplied by the editor after the remainder of the string is sent to provide this delay. The delay can be either an integer, for example, "20", or an integer followed by indicates that the padding required is proportional to the number of lines affected by the operation, and the amount given is the per-affected-unit padding required. When an asterisk (*) is specified, it is sometimes useful to give a delay of the form "3.5" to specify a delay per unit to tenths of milliseconds.

A number of escape sequences are provided in the string-valued capabilities for easy encoding of characters there. A `\E` maps to an ESCAPE character, `O^x` maps to a CTRL/x for any appropriate x, and the sequences `\n \r \t \b \f` give a newline, return, tab, backspace and formfeed. Finally, characters may be given as three octal digits after a backslash (\), and the characters circumflex (^) and backslash (\) may be given as `\^` and `\\`. If it is necessary to place a colon (:) in a capability, it must be escaped in octal as `\072`. If it is necessary to place a null character in a string capability, it must be encoded as `\200`. The routines that deal with `termcap` use C strings and strip the high bits of the output very late so that a `\200` comes out as a `\000` would.

Preparing Descriptions

This section outlines how to prepare descriptions of terminals. The most effective way to prepare a terminal description is by imitating the description of a similar terminal in `termcap` and to build up a description gradually, using partial descriptions with `ex` to check that they are correct. Be aware that a very unusual terminal may expose deficiencies in the ability of the `termcap` file to describe it or bugs in `ex`. To easily test a new terminal description, you can set the environment variable TERMCAP to a pathname of a file containing the description you are working on and the editor will look there rather than in `/etc/termcap`. TERMCAP can also be set to the termcap entry itself to avoid reading the file when starting up the editor. This only works on Version 7 systems.

Basic Capabilities

The number of columns on each line for the terminal is given by the `co` numeric capability. If the terminal is a CRT, the number of lines on the screen is given by the `li` capability. If the terminal wraps around to the beginning of the next line when it reaches the right margin, it should have the `am` capability. If the terminal can clear its screen, this is given by the `cl` string capability. If the terminal can

backspace, it should have the `bs` capability, unless a backspace is accomplished by a character other than ^H, in which case you should give this character as the `bc` string capability. If it overstrikes, rather than clearing a position when a character is struck over, it should have the `os` capability.

A very important point here is that the local cursor motions encoded in `termcap` are undefined at the left and top edges of a CRT terminal. The editor will never attempt to backspace around the left edge, nor will it attempt to go up locally off the top. The editor assumes that feeding off the bottom of the screen will cause the screen to scroll up, and the `am` capability tells whether the cursor sticks at the right edge of the screen. If the terminal has switch-selectable automatic margins, the `termcap` file usually assumes that this is on, that is, `am`.

These capabilities suffice to describe hard-copy and ''glass-tty'' terminals. Thus, the model 33 teletype is described as:

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

The Lear Siegler ADM–3 is described as:

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

Cursor Addressing

Cursor addressing in the terminal is described by a `cm` string capability, with `printf`(3s) types of escapes such as `%x` in it. These substitute to encodings of the current line or column position, while other characters are passed through unchanged. If the `cm` string is thought of as being a function, its arguments are the line and then the column to which motion is desired, and the `%` encodings have the following meanings:

%d	As in *printf*, 0 origin
%2	Like %2d
%3	Like %3d
%.	Like %c
%+x	Adds *x* to value, then %.
%>xy	If value > x adds y, no output.
%r	Reverses order of line and column, no output
%i	Increments line/column (for 1 origin)
%%	Gives a single %
%n	Exclusive or row and column with 0140 (DM2500)
%B	BCD (16*(x/10)) + (x%10), no output.
%D	Reverse coding (x-2*(x%16)), no output. (Delta Data).

Consider the HP2645, which, to get to row 3 and column 12, needs to be sent \E&a12c03Y padded for 6 milliseconds. Note that the order of the rows and columns is inverted here, and that the row and column are printed as 2 digits. Thus, its `cm` capability is ''cm=6\E&%r%2c%2Y''. The Microterm 2ACT-IV needs the current row and column sent preceded by a ^T, with the row and column simply encoded in binary, ''cm=^T%.%.''. Terminals that use ''%.'' need to be able to backspace the cursor (`bs` or `bc`), and to move the cursor up one line on the screen (`up` introduced in the following section). This is necessary because it is not always safe to transmit `\t`, `\n` `^D`, and `\r`, as the system may change or discard them.

A final example is the LSI ADM-3a, which uses row and column offset by a blank character; thus ''cm=\E=%+ %+ ''.

Cursor Motions

If the terminal can move the cursor one position to the right, leaving the character at the current position unchanged, this sequence should be given as `nd` (non-destructive space). If it can move the cursor up a line on the screen in the same column, this should be given as `up`. If the terminal has no cursor addressing capability, but can home the cursor (to very upper left corner of screen), this can be given as `ho`. Similarly a fast way of getting to the lower left hand corner can be given as `ll`. This may involve going up with `up` from the home position, but the editor will never do this itself, unless `ll` does, because it makes no assumption about the effect of moving up from the home position.

Area Clears

If the terminal can clear from the current position to the end of the line, leaving the cursor where it is, this should be given as `ce`. If the terminal can clear from the current position to the end of the display, this should be given as `cd`. The editor only uses `cd` from the first column of a line.

Insert/Delete Line

If the terminal can open a new blank line before the line where the cursor is, this should be given as `al`; this is done only from the first position of a line. The cursor must then appear on the newly blank line. If the terminal can delete the line which the cursor is on, this should be given as `dl`. This is done only from the first position on the line to be deleted. If the terminal can scroll the screen backwards, this can be given as `sb`, but just `al` suffices. If the terminal can retain display memory above, the `da` capability should be given; if display memory can be retained below, `db` should be given. These let the editor understand that deleting a line on the screen may bring nonblank lines up from below or that scrolling back with `sb` may bring down nonblank lines.

Insert/Delete Character

There are two basic kinds of intelligent terminals with respect to the insert/delete character that can be described using `termcap`. The most common insert/delete character operations affect only the characters on the current line and shift characters off the end of the line rigidly. Other terminals, such as the Concept 100 and the Perkin Elmer Owl, make a distinction between typed and untyped blanks on the screen, shifting upon an insert or delete only to an untyped blank on the screen which is either eliminated, or expanded to 2 untyped blanks. You can find out which kind of terminal you have by clearing the screen and typing text separated by cursor motions. Type "abc def" using local cursor motions (not spaces) between the "abc" and the "def". Then, position the cursor before the "abc" and put the terminal in insert mode. If typing characters causes the rest of the line to shift rigidly and characters to fall off the end, your terminal does not distinguish between blanks and untyped positions. If the "abc" shifts over to the "def" and then moves with it around the end of the current line and onto the next line as you insert, you have the second type of terminal, and should give the capability `in`, which stands for "insert null". If your terminal does something different and unusual, you may have to modify the editor to get it to use the insert mode your terminal defines. Virtually all terminals that have an insert mode fall into one of these two classes.

The editor can handle both terminals that have an insert mode and terminals that send a simple sequence to open a blank position on the current line. Give as `im` the sequence to get into insert mode, or give it an empty value if your terminal uses a

sequence to insert a blank position. Give as ei the sequence to leave insert mode (give this with an empty value also, if you gave im so). Give as ic any sequence needed to be sent just before sending the character to be inserted. Most terminals with a true insert mode will not give ic. Terminals that send a sequence to open a screen position should give it here. (Insert mode is preferable to the sequence to open a position on the screen, if your terminal has both.) If post insert padding is needed, give this as a number of milliseconds in ip (a string option). Any other sequence that may need to be sent after an insert of a single character can also be given in ip.

It is occasionally necessary to move around while in insert mode to delete characters on the same line (for example, if there is a tab after the insertion position). If your terminal allows motion while in insert mode, you can give the capability mi to speed up inserting in this case. Omitting mi affects only speed. Some terminals (notably Datamedia's) must not have mi because of the way their insert mode works.

Finally, you can specify delete mode by giving dm and ed to enter and exit delete mode, and dc to delete a single character while in delete mode.

Highlighting, Underlining, and Visible Bells

If your terminal has sequences to enter and exit standout mode, these can be given as so and se, respectively. If there are several kinds of standout mode, such as inverse video, blinking, or underlining. Half-bright is not usually an acceptable "standout" mode, unless the terminal is in inverse video mode constantly. The preferred mode is inverse video by itself. If the code to change into or out of standout mode leaves 1 or even 2 blank spaces on the screen, as the TVI 912 and Teleray 1061 do, ug should be given to tell how many spaces are left.

Codes to begin underlining and end underlining can be given as us and ue, respectively. If the terminal has a code to underline the current character and move the cursor one space to the right, such as the Microterm Mime, this can be given as uc. (If the underline code does not move the cursor to the right, give the code followed by a nondestructive space.)

Many terminals, such as the HP 2621, automatically leave standout mode when they move to a new line or the cursor is addressed. Programs using standout mode should exit standout mode before moving the cursor or sending a newline.

If the terminal has a way of flashing the screen to indicate an error quietly (a bell replacement), this can be given as vb; it must not move the cursor. If the terminal should be placed in a different mode during open and visual modes of ex, this can be given as vs and ve, sent at the start and end of these modes respectively. These can be used to change, for example, from an 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, the codes to enter and exit this mode can be given as ti and te. This arises, for example, from terminals like the Concept with more than one page of memory. If the terminal has only memory-relative cursor addressing and not screen-relative cursor addressing, a one-screen sized window must be fixed into the terminal for cursor addressing to work properly.

If your terminal correctly generates underlined characters, with no special codes needed, even though it does not overstrike, you should give the capability ul. If overstrikes are erasable with a blank, this should be indicated by giving eo.

Keypad

If the terminal has a keypad that transmits codes when the keys are pressed, this information can be given. Note that it is not possible to handle terminals where the keypad only works in local (this applies, for example, to the unshifted HP 2621 keys). If the keypad can be set to transmit or not transmit, give these codes as `ks` and `ke`. Otherwise, the keypad is assumed to always transmit. The codes sent by the left arrow, right arrow, up arrow, down arrow, and home keys can be given as `kl`, `kr`, `ku`, `kd`, and `kh`, respectively. If there are function keys such as f0, f1, ..., f9, the codes they send can be given as `k0`, `k1`, ..., `k9`. If these keys have labels other than the default f0 through f9, the labels can be given as `l0`, `l1`, ..., `l9`. If there are other keys that transmit the same code as the terminal expects for the corresponding function, such as clear screen, the `termcapP` 2-letter codes can be given in the `ko` capability, for example, ":ko=cl,ll,sf,sb:", which 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` entries.

The `ma` entry is also used to indicate arrow keys on terminals that have single-character arrow keys. It is obsolete, but still in use in Version 2 of `vi`, which must be run on some minicomputers due to memory limitations. This field is redundant with `kl`, `kr`, `ku`, `kd`, and `kh`. It consists of groups of 2 characters. In each group, the first character is what an arrow key sends; the second character is the corresponding `vi` command. These commands are `h` for `kl`, `j` for `kd`, `k` for `ku`, `l` for `kr`, and `H` for `kh`. For example, the Microterm Mime would be ":ma=^Kj^Zk^Xl:" indicating arrow keys left (^H), down (^K), up (^Z), and right (^X). (There is no home key on the Mime.)

Miscellaneous

If the terminal requires other than a null (zero) character as a pad, this can be given as `pc`.

If tabs on the terminal require padding, or if the terminal uses a character other than `^I` to tab, this can be given as `ta`.

Hazeltine terminals, which do not allow tildes (~) to be printed, should indicate `hz`. Datamedia terminals, which echo carriage-return linefeed for a carriage return and then ignore a following linefeed, should indicate `nc`. Early Concept terminals, which ignore a linefeed immediately after an `am` wrap, should indicate `xn`. If an erase-eol is required to get rid of standout (instead of merely writing on top of it), `xs` should be given. Teleray terminals, where tabs turn all characters moved over to blanks, should indicate `xt`. Other specific terminal problems may be corrected by adding more capabilities of the form *xx*.

Other capabilities include `is`, an initialization string for the terminal, and `if`, the name of a file containing long initialization strings. These strings are expected to properly clear and then set the tabs on the terminal, if the terminal has settable tabs. If both are given, `is` is printed before `if`. This is useful where `if` is `/usr/lib/tabset/std` but `is` clears the tabs first.

Similar Terminals

If there are two very similar terminals, one can be defined as being just like the other with certain exceptions. The string capability `tc` can be given with the name of the similar terminal. This capability must be *last* and the combined length of the two entries must not exceed 1024. Since `termlib` routines search the entry from left to right, and since the tc capability is replaced by the corresponding entry, the

capabilities given at the left override the ones in the similar terminal. A capability can be canceled with `xx@`, where xx is the capability. For example, the following entry defines a 2621nl that does not have the `ks` or `ke` capabilities, and hence does not turn on the function key labels when in visual mode:

```
hn | 2621nl:ks@:ke@:tc=2621:
```

This is useful for different modes for a terminal or for different user preferences.

Restrictions

The `ex` command allows only 256 characters for string capabilities, and the routines in `termcap`(3x) do not check for overflow of this buffer. The total length of a single entry (excluding only escaped newlines) cannot exceed 1024.

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

Files

`/etc/termcap`	File containing terminal descriptions

See Also

ex(1), more(1), tset(1), ul(1), vi(1), curses(3x), termcap(3x)

Name

terminfo – terminal capability database

Syntax

/usr/lib/terminfo/*/*

Description

The `terminfo` database describes terminals by giving a set of capabilities which the terminals have, and by describing how the operations are performed by the terminals. Padding requirements and initialization sequences are included in `terminfo`.

Entries in `terminfo` consist of a number of fields separated by commas (,) . White space after each comma (,) is ignored. The first entry for each terminal provides the known name of the terminal, separated by vertical bars (|). The first name given is the most common abbreviation for the terminal; the last name given should be a long name fully identifying the terminal. All others are understood as synonyms for the terminal name. All names, with the exception of the last, should be in lowercase and cannot contain blanks; the last name can contain uppercase characters and blanks for readability.

Terminal names, except for the last, should be chosen using the following conventions:

- The piece of hardware that makes up the terminal should have a root name chosen. For example, hp2621.
- The root name cannot contain hyphens, but synonyms can be used that do not conflict with other names.
- Modes that the hardware can be in, or user preferences, should be indicated by appending a hyphen and an indicator of the mode. For example, a VT100 in 132 column mode would be vt100-w.
- The following suffixes should be used where possible:

Suffix	Meaning	Example
-w	Wide mode (more than 80 columns)	vt100-w
-am	With auto. margins (usually default)	vt100-am
-nam	Without automatic margins	vt100-nam
-*n*	Number of lines on the screen	aaa-60
-na	No arrow keys (leave them in local)	c100-na
-*n*p	Number of pages of memory	c100-4p
-rv	Reverse video	c100-rv

The following headers are used in the capabilities table:

Variable booleans Variable is the name by which the programmer (at the terminfo level) accesses the capability.

Capname Short name used in the text of the database, and is used by a person updating the database.

I.code Two-letter internal code used in the compiled database, which always corresponds to the old `termcap` capability name.

Capability names have no hard length limit, but an informal limit of 5 characters has been adopted to keep them short and to allow the tabs in the source file caps to line up nicely. Whenever possible, names are chosen to be the same as, or similar to, the ANSI X3.64-1979 standard. Semantics are also intended to match those of the specification. They are as follows:

(P) Indicates that padding may be specified.

(G) Indicates that the string is passed through tparm with parms as given (#*i*).

(*) Indicates that padding may be based on the number of lines affected

($\#_i$) Indicates the i^{th} parameter.

Variable Booleans	Cap-name	I. Code	Description
auto_left_margin,	bw	bw	cub1 wraps from column 0 to last column
auto_right_margin,	am	am	Terminal has automatic margins
beehive_glitch,	xsb	xb	Beehive (f1=escape, f2=ctrl C)
ceol_standout_glitch,	xhp	xs	Standout not erased by over-writing (hp)
eat_newline_glitch,	xenl	xn	Newline ignored after 80 cols (Concept)
erase_overstrike,	eo	eo	Can erase overstrikes with a blank
generic_type,	gn	gn	Generic line type (for ex., dialup, switch).
hard_copy,	hc	hc	Hardcopy terminal
has_meta_key,	km	km	Has a meta key (shift, sets parity bit)
has_status_line,	hs	hs	Has extra status line
insert_null_glitch,	in	in	Insert mode distinguishes nulls
memory_above,	da	da	Display may be retained above the screen
memory_below,	db	db	Display may be retained below the screen
move_insert_mode,	mir	mi	Safe to move while in insert mode
move_standout_mode,	msgr	ms	Safe to move in standout modes
over_strike,	os	os	Terminal overstrikes
status_line_esc_ok,	eslok	es	Escape can be used on the status line
teleray_glitch,	xt	xt	Tabs ruin, magic so char (Teleray 1061)
tilde_glitch,	hz	hz	Hazeltine; can not print tildes (~)s
transparent_underline,	ul	ul	underline character overstrikes
xon_xoff,	xon	xo	Terminal uses xon/xoff handshaking
Numbers:			
columns,	cols	co	Number of columns in a line
init_tabs,	it	it	Tabs initially every # spaces
lines,	lines	li	Number of lines on screen or page
lines_of_memory,	lm	lm	Lines of memory if > lines. 0

			means varies
magic_cookie_glitch,	xmc	sg	Number of blank chars left by smso or rmso
padding_baud_rate,	pb	pb	Lowest baud where cr/nl padding is needed
virtual_terminal,	vt	vt	Virtual terminal number (UNIX system)
width_status_line,	wsl	ws	Number of columns in status line
Strings:			
back_tab,	cbt	bt	Back tab (P)
bell,	bel	bl	Audible signal (bell) (P)
carriage_return,	cr	cr	Carriage return (P*)
change_scroll_region,	csr	cs	Change to lines #1 through #2 (vt100) (PG)
clear_all_tabs,	tbc	ct	Clear all tab stops (P)
clear_screen,	clear	cl	Clear screen and home cursor (P*)
clr_eol,	el	ce	Clear to end of line (P)
clr_eos,	ed	cd	Clear to end of display (P*)
column_address,	hpa	ch	Set cursor column (PG)
command_character,	cmdch	CC	Term. settable cmd char in prototype
cursor_address,	cup	cm	Screen rel. cursor motion row #1 col #2 (PG)
cursor_down,	cud1	do	Down one line
cursor_home,	home	ho	Home cursor (if no cup)
cursor_invisible,	civis	vi	Make cursor invisible
cursor_left,	cub1	le	Move cursor left one space
cursor_mem_address,	mrcup	CM	Memory relative cursor addressing
cursor_normal,	cnorm	ve	Make cursor appear normal (undo vs/vi)
cursor_right,	cuf1	nd	Nondestructive space (cursor right)
cursor_to_ll,	ll	ll	Last line, first column (if no cup)
cursor_up,	cuu1	up	Upline (cursor up)
cursor_visible,	cvvis	vs	Make cursor very visible
delete_character,	dch1	dc	Delete character (P*)
delete_line,	dl1	dl	Delete line (P*)
dis_status_line,	dsl	ds	Disable status line
down_half_line,	hd	hd	Half-line down (forward 1/2 linefeed)
enter_alt_charset_mode,	smacs	as	Start alternate character set (P)
enter_blink_mode,	blink	mb	Turn on blinking
enter_bold_mode,	bold	md	Turn on bold (extra bright) mode
enter_ca_mode,	smcup	ti	String to begin programs that use cup
enter_delete_mode,	smdc	dm	Delete mode (enter)
enter_dim_mode,	dim	mh	Turn on half-bright mode
enter_insert_mode,	smir	im	Insert mode (enter);
enter_protected_mode,	prot	mp	Turn on protected mode
enter_reverse_mode,	rev	mr	Turn on reverse video mode
enter_secure_mode,	invis	mk	Turn on blank mode (chars invisible)

enter_standout_mode,	smso	so	Begin stand out mode
enter_underline_mode,	smul	us	Start underscore mode
erase_chars	ech	ec	Erase #1 characters (PG)
exit_alt_charset_mode,	rmacs	ae	End alternate character set (P)
exit_attribute_mode,	sgr0	me	Turn off all attributes
exit_ca_mode,	rmcup	te	String to end programs that use cup
exit_delete_mode,	rmdc	ed	End delete mode
exit_insert_mode,	rmir	ei	End insert mode
exit_standout_mode,	rmso	se	End stand out mode
exit_underline_mode,	rmul	ue	End underscore mode
flash_screen,	flash	vb	Visible bell (may not move cursor)
form_feed,	ff	ff	Hardcopy terminal page eject (P*)
from_status_line,	fsl	fs	Return from status line
init_1string,	is1	i1	Terminal initialization string
init_2string,	is2	i2	Terminal initialization string
init_3string,	is3	i3	Terminal initialization string
init_file,	if	if	Name of file containing is
insert_character,	ich1	ic	Insert character (P)
insert_line,	il1	al	Add new blank line (P*)
insert_padding,	ip	ip	Insert pad after character inserted (p*)
key_backspace,	kbs	kb	Sent by backspace key
key_catab,	ktbc	ka	Sent by clear-all-tabs key
key_clear,	kclr	kC	Sent by clear screen or erase key
key_ctab,	kctab	kt	Sent by clear-tab key
key_dc,	kdch1	kD	Sent by delete character key
key_dl,	kdl1	kL	Sent by delete line key
key_down,	kcud1	kd	Sent by terminal down arrow key
key_eic,	krmir	kM	Sent by rmir or smir in insert mode
key_eol,	kel	kE	Sent by clear-to-end-of-line key
key_eos,	ked	kS	Sent by clear-to-end-of-screen key
key_f0,	kf0	k0	Sent by function key f0
key_f1,	kf1	k1	Sent by function key f1
key_f10,	kf10	ka	Sent by function key f10
key_f2,	kf2	k2	Sent by function key f2
key_f3,	kf3	k3	Sent by function key f3
key_f4,	kf4	k4	Sent by function key f4
key_f5,	kf5	k5	Sent by function key f5
key_f6,	kf6	k6	Sent by function key f6
key_f7,	kf7	k7	Sent by function key f7
key_f8,	kf8	k8	Sent by function key f8
key_f9,	kf9	k9	Sent by function key f9
key_home,	khome	kh	Sent by home key
key_ic,	kich1	kI	Sent by ins char/enter ins mode key
key_il,	kil1	kA	Sent by insert line
key_left,	kcub1	kl	Sent by terminal left arrow key
key_ll,	kll	kH	Sent by home-down key
key_npage,	knp	kN	Sent by next-page key
key_ppage,	kpp	kP	Sent by previous-page key

key_right,	kcuf1	kr	Sent by terminal right arrow key
key_sf,	kind	kF	Sent by scroll-forward/down key
key_sr,	kri	kR	Sent by scroll-backward/up key
key_stab,	khts	kT	Sent by set-tab key
key_up,	kcuu1	ku	Sent by terminal up arrow key
keypad_local,	rmkx	ke	Out of "keypad transmit" mode
keypad_xmit,	smkx	ks	Put terminal in "keypad transmit" mode
lab_f0,	lf0	l0	Labels on function key f0 if not f0
lab_f1,	lf1	l1	Labels on function key f1 if not f1
lab_f10,	lf10	la	Labels on function key f10 if not f10
lab_f2,	lf2	l2	Labels on function key f2 if not f2
lab_f3,	lf3	l3	Labels on function key f3 if not f3
lab_f4,	lf4	l4	Labels on function key f4 if not f4
lab_f5,	lf5	l5	Labels on function key f5 if not f5
lab_f6,	lf6	l6	Labels on function key f6 if not f6
lab_f7,	lf7	l7	Labels on function key f7 if not f7
lab_f8,	lf8	l8	Labels on function key f8 if not f8
lab_f9,	lf9	l9	Labels on function key f9 if not f9
meta_on,	smm	mm	Turn on "meta mode" (8th bit)
meta_off,	rmm	mo	Turn off "meta mode"
newline,	nel	nw	Newline (behaves like cr followed by lf)
pad_char,	pad	pc	Pad character (rather than null)
parm_dch,	dch	DC	Delete #1 chars (PG*)
parm_delete_line,	dl	DL	Delete #1 lines (PG*)
parm_down_cursor,	cud	DO	Move cursor down #1 lines (PG*)
parm_ich,	ich	IC	Insert #1 blank chars (PG*)
parm_index,	indn	SF	Scroll forward #1 lines (PG)
parm_insert_line,	il	AL	Add #1 new blank lines (PG*)
parm_left_cursor,	cub	LE	Move cursor left #1 spaces (PG)
parm_right_cursor,	cuf	RI	Move cursor right #1 spaces (PG*)
parm_rindex,	rin	SR	Scroll backward #1 lines (PG)
parm_up_cursor,	cuu	UP	Move cursor up #1 lines (PG*)
pkey_key,	pfkey	pk	Prog funct key #1 to type string #2
pkey_local,	pfloc	pl	Prog funct key #1 to execute string #2
pkey_xmit,	pfx	px	Prog funct key #1 to xmit string #2
print_screen,	mc0	ps	Print contents of the screen
prtr_off,	mc4	pf	Turn off the printer
prtr_on,	mc5	po	Turn on the printer
repeat_char,	rep	rp	Repeat char #1 #2 times. (PG*)
reset_1string,	rs1	r1	Reset terminal completely to sane modes.
reset_2string,	rs2	r2	Reset terminal completely to sane modes.
reset_3string,	rs3	r3	Reset terminal completely to sane modes.
reset_file,	rf	rf	Name of file containing reset string
restore_cursor,	rc	rc	Restore cursor to position of

			last sc
row_address,	vpa	cv	Vertical position absolute (set row) (PG)
save_cursor,	sc	sc	Save cursor position (P)
scroll_forward,	ind	sf	Scroll text up (P)
scroll_reverse,	ri	sr	Scroll text down (P)
set_attributes,	sgr	sa	Define the video attributes (PG9)
set_tab,	hts	st	Set a tab in all rows, current column
set_window,	wind	wi	Current window is lines #1-#2 cols #3-#4
tab,	ht	ta	Tab to next 8 space hardware tab stop
to_status_line,	tsl	ts	Go to status line, column #1
underline_char,	uc	uc	Underscore one char and move past it
up_half_line,	hu	hu	Half-line up (reverse 1/2 linefeed)
init_prog,	iprog	iP	Path name of program for init
key_a1,	ka1	K1	Upper left of keypad
key_a3,	ka3	K3	Upper right of keypad
key_b2,	kb2	K2	Center of keypad
key_c1,	kc1	K4	Lower left of keypad
key_c3,	kc3	K5	Lower right of keypad
prtr_non,	mc5p	pO	Turn on the printer for #1 bytes

Sample Entry

The following entry, which describes the Concept–100, is one of the more complex entries in the `terminfo`:

```
concept100|c100|concept|c104|c100-4p|concept 100,
   am, bel=^G, blank=\EH, blink=\EC, clear=^L$<2*>, cnorm=\Ew,
   cols#80, cr=^M$<9>, cub1=^H, cud1=^J, cuf1=\E=,
   cup=\Ea%p1%' '%+%c%p2%' '%+%c,
   cuu1=\E;, cvvis=\EW, db, dch1=\E^A$<16*>, dim=\EE, dl1=\E^B$<3*>,
   ed=\E^C$<16*>, el=\E^U$<16>, eo, flash=\Ek$<20>\EK, ht=\t$<8>,
   ill=\E^R$<3*>, in, ind=^J, .ind=^J$<9>, ip=$<16*>,
   is2=\EU\Ef\E7\E5\E8\El\ENH\EK\E\200\Eo&\200\Eo\47\E,
   kbs=^h, kcub1=\E>, kcud1=\E<, kcuf1=\E=, kcuu1=\E;,
   kf1=\E5, kf2=\E6, kf3=\E7, khome=\E?,
   lines#24, mir, pb#9600, prot=\EI, rep=\Er%p1%c%p2%' '%+%c$<.2*>,
   rev=\ED, rmcup=\Ev    $<6>\Ep\r\n, rmir=\E\200, rmkx=\Ex,
   rmso=\Ed\Ee, rmul=\Eg, rmul=\Eg, sgr0=\EN\200,
   smcup=\EU\Ev  8p\Ep\r, smir=\E^P, smkx=\EX, smso=\EE\ED,
   smul=\EG, tabs, ul, vt#8, xenl,
```

Entries can continue onto multiple lines by placing white space at the beginning of each line, with the exception of the first line. Comments can be included, as long as the comment is preceded by a number sign (#). The following list describes terminal capabilities in more detail.

Types of Capabilities

Boolean capabilities Indicate that the terminal has some particular feature. For example, the Concept–100 has automatic margins (an automatic return and linefeed when the end-of-line is reached). This is described in the Boolean capabilities column as an **am.**

Numeric capabilities Provide the size of the terminal or the size of particular delays. Numeric capabilities are followed by a number sign (#) and then the value. Hence, the **cols**, which indicates the number of column the terminal has, provides the value 80 for the Concept.

String capabilities Provide a sequence that can be used to perform particular terminal operations. Hence, string-valued capabilities such as **el** (clear to the end-of-line sequence are described the 2-character code (an equal sign (=) and then a string ending at the next comma (,). A delay in milliseconds can appear anywhere in such a capability, enclosed in $<..> brackets, and padding characters are supplied by `tputs` to provide this delay.

A delay can be either a number, such as 20, or a number followed by an asterisk (*), such as 3*. The asterisk (*) indicates that the padding required is proportional to the number of lines affected by the operation, and the amount given is the per-affected-unit padding required. (In the case of the insert character, the factor is still the number of lines affected. This is always one, unless the terminal has **xenl** and the software uses it.) When an asterisk (*) is specified, it is sometimes useful to give a delay of the form 3.5, which indicates a delay per unit to tenths of milliseconds. (Only one decimal place is allowed.)

Escape sequences are provided in the string-valued capabilities for easy encoding of characters there. Both \E and \e map to an ESCAPE character, **^x** maps to a control-x for any appropriate x, and the sequences **\n \l \r \t \b \f \s** give a newline, linefeed, return, tab, backspace, formfeed, and space. Other escapes include \^ for ^, \\ for \, \, for comma, \: for :, and \0 for null. (\0 will produce \200, which does not terminate a string but behaves as a null character on most terminals.) Finally, characters may be given as three octal digits after a \.

Sometimes individual capabilities must be commented out. To do this, put a period before the capability name. For example, see the second **ind** in the previous Sample Entry.

Preparing Descriptions

This section describes how to prepare a description of a terminal. The most effective way to prepare a terminal description is by imitating the description of a similar terminal in `terminfo` and to build up a description gradually, using partial descriptions with `vi` to ensure that they are correct. A very unusual terminal or errors in `vi` may expose deficiencies in the ability of the `terminfo` file to describe it.

To test a new terminal description, set the environment variable TERMINFO to a pathname of a directory containing the compiled description you are working on. The programs can search this directory rather than search */usr/lib/terminfo*. To get the padding for insert line right (if the terminal manufacturer did not document it), edit /etc/passwd at 9600 baud, delete 16 or so lines from the middle of the screen, then type the character u several times quickly. If the terminal behaves erratically, more padding is usually needed. A similar test can be used for the insert character.

Basic Capabilities

The number of columns on each line for the terminal is specified by the **cols** numeric capability. If the terminal is a CRT, then the number of lines on the screen is given by the **lines** capability. If the terminal wraps around to the beginning of the next line when it reaches the right margin, then it should have the **am** capability. If the terminal can clear its screen, leaving the cursor in the home position, then this is given by the **clear** string capability. If the terminal overstrikes (rather than clearing a position when a character is struck over), then it should have the **os** capability. If the terminal is a printing terminal, with no soft copy unit, give it both **hc** and **os**. (**os** applies to storage scope terminals, such as TEKTRONIX 4010 series, as well as hard copy and APL terminals.) If there is a code to move the cursor to the left edge of the current row, give this as **cr**. (Normally this will be carriage return, control M.) If there is a code to produce an audible signal (bell, beep, etc), give this as **bel**.

If there is a code to move the cursor one position to the left (such as backspace) that capability should be given as **cub1**. Similarly, codes to move to the right, up, and down should be given as **cuf1**, **cuu1**, and **cud1**. These local cursor motions should not alter the text they pass over; for example, you would not normally use '**cuf1**= ' because the space would erase the character moved over.

NOTE

The local cursor motions encoded in `terminfo` are undefined at the left and top edges of a CRT terminal. Programs should never attempt to backspace around the left edge, unless **bw** is given, and never attempt to go up locally off the top. In order to scroll text up, a program will go to the bottom left corner of the screen and send the **ind** (index) string.

To scroll text down, a program goes to the top left corner of the screen and sends the **ri** (reverse index) string. The strings **ind** and **ri** are undefined when not on their respective corners of the screen.

Parameterized versions of the scrolling sequences are **indn** and **rin**, which have the same semantics as **ind** and **ri**, except that they take one parameter and scroll that many lines. They are also undefined, except at the appropriate edge of the screen.

The **am** capability tells whether the cursor sticks at the right edge of the screen when text is output, but this does not necessarily apply to a **cuf1** from the last column. The only local motion that is defined from the left edge is if **bw** is given, then a **cub1** from the left edge will move to the right edge of the previous row. If **bw** is not given, the effect is undefined. This is useful for drawing a box around the edge of the screen, for example. If the terminal has switch-selectable automatic margins, the `terminfo` file usually assumes that this is on; that is, **am**. If the terminal has a command which moves to the first column of the next line, that command can be given as **nel** (newline). It does not matter if the command clears the remainder of the current line, so, if the terminal has no **cr** and **lf**, it may still be possible to craft a working **nel** out of one or both of them.

These capabilities suffice to describe hardcopy and glass-tty terminals. Thus, the Model 33 Teletype is described as:

```
33|tty33|tty|model 33 teletype,
bel=^G, cols#72, cr=^M, cud1=^J, hc, ind=^J, os,
```

The Lear Siegler ADM–3 is described as:

```
adm3|3|lsi adm3,
```

```
am, bel=^G, clear=^Z, cols#80, cr=^M, cub1=^H, cud1=^J,
ind=^J, lines#24,
```

Parameterized Strings

Cursor addressing and other strings requiring parameters in the terminal are described by a parameterized string capability, with `printf`(3s), such as escapes like **%x**. For example, to address the cursor, the **cup** capability is given, using two parameters: the row and column to address to. (Rows and columns are numbered from zero and refer to the physical screen visible to the user, not to any unseen memory.) If the terminal has memory-relative cursor addressing, that can be indicated by **mrcup**.

The parameter mechanism uses a stack and special **%** codes to manipulate it. Typically a sequence pushes one of the parameters onto the stack and then prints it in some format. Often, more complex operations are necessary.

The percent sign (**%**) encodings have the following meanings:

%%	outputs '%'
%d	print pop() as in printf
%2d	print pop() like %2d
%3d	print pop() like %3d
%02d	
%03d	as in printf
%c	print pop() gives %c
%s	print pop() gives %s
%p[1-9]	push ith parm
%P[a-z]	set variable [a-z] to pop()
%g[a-z]	get variable [a-z] and push it
%'c'	char constant c
%{nn}	integer constant nn
%+ %- %* %/ %m	arithmetic (%m is mod): push(pop() op pop())
%& %| %^	bit operations: push(pop() op pop())
%= %> %<	logical operations: push(pop() op pop())
%! %~	unary operations push(op pop())
%i	add 1 to first two parms (for ANSI terminals)
%? expr %t thenpart %e elsepart %;	if-then-else, %e elsepart is optional. else-if's are possible ala Algol 68: %? c_1 %t b_1 %e c_2 %t b_2 %e c_3 %t b_3 %e c_4 %t b_4 %e %; c_i are conditions, b_i are bodies.

Binary operations are in postfix form with the operands in the usual order. That is, to get x-5, use %gx%{5}%-.

Consider the HP2645, which, to get to row 3 and column 12, needs to be sent \E&a12c03Y padded for 6 milliseconds. Note that the order of the rows and columns is inverted here, and that the row and column are printed as two digits. Thus, its **cup** capability is cup=6\E&%p2%2dc%p1%2dY.

The Microterm ACT-IV needs the current row and column sent preceded by a **^T**, with the row and column simply encoded in binary, cup=^T%p1%c%p2%c. Terminals that use %c need to be able to backspace the cursor (**cub1**), and to move

the cursor up one line on the screen (**cuu1**). This is necessary because it is not always safe to transmit **\n ^D** and **\r**, as the system may change or discard them. (The library routines dealing with terminfo set tty modes so that tabs are never expanded, so \t is safe to send. This turns out to be essential for the Ann Arbor 4080.)

A final example is the LSI ADM-3a, which uses row and column offset by a blank character; thus cup=\E=%p1%' '%+%c%p2%' '%+%. After sending \E=, this pushes the first parameter, pushes the ASCII value for a space (32), adds them (pushing the sum on the stack in place of the two previous values), and outputs that value as a character. Then, the same is done for the second parameter. More complex arithmetic is possible using the stack.

If the terminal has row or column absolute cursor addressing, these can be given as single parameter capabilities **hpa** (horizontal position absolute) and **vpa** (vertical position absolute). Sometimes, these are shorter than the more general 2-parameter sequence (as with the hp2645) and can be used in preference to **cup** . If there are parameterized local motions (for example, move *n* spaces to the right), these can be given as **cud**, **cub**, **cuf**, and **cuu**, with a single parameter indicating how many spaces to move. These are primarily useful if the terminal does not have **cup**, such as the TEKTRONIX 4025.

Cursor Motions

If the terminal has a fast way to home the cursor (to very upper left corner of screen), then this can be given as **home**. Similarly, a fast way of getting to the lower left-hand corner can be given as **ll**. This may involve going up with **cuu1** from the home position, but a program should never do this itself (unless **ll** does), because it can make no assumption about the effect of moving up from the home position. Note that the home position is the same as addressing to (0,0): the top left corner of the screen, not memory. Thus, the \EH sequence on HP terminals cannot be used for **home**.

Area Clears

If the terminal can clear from the current position to the end of the line, leaving the cursor where it is, this should be given as **el**. If the terminal can clear from the current position to the end of the display, this should be given as **ed**. **Ed** is only defined from the first column of a line. Thus, it can be simulated by a request to delete a large number of lines, if a true **ed** is not available.

Insert/delete line

If the terminal can open a new blank line before the line where the cursor is, this should be given as **il1**; this is done only from the first position of a line. The cursor must then appear on the newly blank line. If the terminal can delete the line that the cursor is on, this should be given as **dl1**; this is done only from the first position on the line to be deleted. Versions of **il1** and **dl1** that take a single parameter and insert or delete that many lines can be given as **il** and **dl**. If the terminal has a settable scrolling region (like the VT100), the command to set this can be described with the **csr** capability, which takes two parameters: the top and bottom lines of the scrolling region. The cursor position is undefined after using this command. It is possible to get the effect of insert or delete line using this command. The **sc** and **rc** (save and restore cursor) commands are also useful. Inserting lines at the top or bottom of the screen can also be done using **ri** or **ind** on many terminals without a true insert/delete line, and this is often faster even on terminals with those features.

If the terminal has the ability to define a window as part of memory, which all commands affect, it should be given as the parameterized string **wind**. The four parameters are the starting and ending lines in memory and the starting and ending columns in memory, in that order.

If the terminal can retain display memory above, then the **da** capability should be given; if display memory can be retained below, then **db** should be given. These indicate that deleting a line or scrolling may bring nonblank lines up from below or that scrolling back with **ri** may bring down nonblank lines.

Insert/Delete Character

There are two basic kinds of intelligent terminals with respect to insert/delete character that can be described using `terminfo`. The most common insert/delete character operations affect only the characters on the current line and shift characters off the end of the line rigidly. Other terminals, such as the Concept-100 and the Perkin Elmer Owl, make a distinction between typed and untyped blanks on the screen, shifting upon an insert or delete only to an untyped blank on the screen that is either eliminated or expanded to two untyped blanks. You can determine the kind of terminal you have by clearing the screen and typing text separated by cursor motions. Type "abc def", using local cursor motions (not spaces) between the abc and the def. Then, position the cursor before the abc and put the terminal in insert mode. If typing characters causes the rest of the line to shift rigidly and characters fall off the end, then your terminal does not distinguish between blanks and untyped positions. If the abc shifts over to the def which then move together around the end of the current line and onto the next as you insert, you have the second type of terminal. You should give the capability **in**, which stands for "insert null". While these are two logically separate attributes (one line, as opposed to multiline insert mode, and special treatment of untyped spaces) every terminal's insert mode can be described with the single attribute.

Terminfo can describe both terminals that have an insert mode and terminals that send a simple sequence to open a blank position on the current line. Give as **smir** the sequence to get into insert mode. Give as **rmir** the sequence to leave insert mode. Then, give as **ich1** any sequence needed to be sent just before sending the character to be inserted. Most terminals with a true insert mode will not give **ich1**; terminals that send a sequence to open a screen position should give it here.

NOTE

> If your terminal has both, insert mode is usually preferable to **ich1**. Do not give both, unless the terminal actually requires both to be used in combination.

If post insert padding is needed, give this as a number of milliseconds in **ip** (a string option). Any other sequence that may need to be sent after an insert of a single character may also be given in **ip**. If your terminal needs both to be placed into an insert mode and a special code to precede each inserted character, both **smir/rmir** and **ich1** can be given, and both will be used. The **ich** capability, with one parameter, *n*, will repeat the effects of **ich1** *n* times.

It is occasionally necessary to move around while in insert mode to delete characters on the same line (for example, if there is a tab after the insertion position). If your terminal allows motion while in insert mode, you can give the capability **mir** to speed up inserting. Omitting **mir** affects only speed. Some terminals (notably Datamedia's) must not have **mir** because of the way their insert mode works.

Finally, you can specify **dch1** to delete a single character, **dch**, with one parameter, *n*, to delete *n characters,* and delete mode by giving **smdc** and **rmdc** to enter and exit delete mode (any mode the terminal needs to be placed in for **dch1** to work).

A command to erase *n* characters (equivalent to outputting *n* blanks, without moving the cursor) can be given as **ech** with one parameter.

Highlighting, Underlining, and Visible Bells

If your terminal has one or more kinds of display attributes, these can be represented in a number of different ways. You should choose one display form as standout mode, representing a good, high contrast, easy to read, format for highlighting error messages and other important information. If you have a choice, reverse video plus half-bright is good, or reverse video alone. The sequences to enter and exit standout mode are given as **smso** and **rmso**, respectively. If the code to change into or out of standout mode leaves one or even two blank spaces on the screen, as the TVI 912 and Teleray 1061 do, then **xmc** should be given to tell how many spaces are left.

Codes to begin underlining and end underlining can be given as **smul** and **rmul**, respectively. If the terminal has a code to underline the current character and move the cursor one space to the right, such as the Microterm Mime, this can be given as **uc**.

Other capabilities to enter various highlighting modes include **blink** (blinking), **bold** (bold or extra bright), **dim** (dim or half-bright), **invis** (blanking or invisible text), **prot** (protected), **rev** (reverse video), **sgr0** (turn off *all* attribute modes), **smacs** (enter alternate character set mode), and **rmacs** (exit alternate character set mode). Turning on any of these modes singly may or may not turn off other modes.

If there is a sequence to set arbitrary combinations of modes, this should be given as **sgr** (set attributes), taking nine parameters. Each parameter is either 0 or 1, as the corresponding attribute is on or off. The nine parameters are, in order: standout, underline, reverse, blink, dim, bold, blank, protect, and alternate character set. Not all modes need be supported by **sgr**, only those for which corresponding separate attribute commands exist.

Terminals with the "magic cookie glitch" (**xmc**) deposit special cookies when they receive mode-setting sequences, which affect the display algorithm rather than having extra bits for each character. Some terminals, such as the HP 2621, automatically leave standout mode when they move to a new line or the cursor is addressed. Programs using standout mode should exit standout mode before moving the cursor or sending a newline, unless the **msgr** capability, asserting that it is safe to move in standout mode, is present.

If the terminal has a way of flashing the screen to indicate an error quietly (a bell replacement), this can be given as **flash**; however, it must not move the cursor.

If the cursor needs to be made more visible than normal when it is not on the bottom line (to make, for example, a non-blinking underline into an easier to find block or blinking underline), give this sequence as **cvvis**. If you wish to make the cursor completely invisible, give that as **civis**. The capability **cnorm** should be given which undoes the effects of both of these modes.

If the terminal needs to be in a special mode when running a program that uses these capabilities, the codes to enter and exit this mode can be given as **smcup** and **rmcup**. This arises, for example, from terminals like the Concept-100 with more than one page of memory. If the terminal has only memory-relative cursor addressing and not

screen-relative cursor addressing, a one screen-sized window must be fixed into the terminal for cursor addressing to work properly. This is also used for the TEKTRONIX 4025, where **smcup** sets the command character to be the one used by terminfo.

If your terminal correctly generates underlined characters (with no special codes needed) even though it does not overstrike, you should give the capability **ul**. If overstrikes are erasable with a blank, this should be indicated by giving **eo**.

Keypad

If the terminal has a keypad that transmits codes when the keys are pressed, give this information. Note that it is not possible to handle terminals where the keypad only works in local (this applies, for example, to the unshifted HP 2621 keys). If the keypad can be set to transmit or not transmit, give these codes as **smkx** and **rmkx**. Otherwise, the keypad is always assumed to transmit. The codes sent by the left arrow, right arrow, up arrow, down arrow, and home keys can be given as **kcub1, kcuf1, kcuu1, kcud1,** and **khome**, respectively. If there are function keys such as f0, f1, ... f10, the codes they send can be given as **kf0, kf1, ... kf10**. If these keys have labels other than the default f0 through f10, the labels can be given as **lf0, lf1, ... lf10**. The codes transmitted by certain other special keys can be given: **kll** (home down), **kbs** (backspace), **ktbc** (clear all tabs), **kctab** (clear the tab stop in this column), **kclr** (clear screen or erase key), **kdch1** (delete character), **kdl1** (delete line), **krmir** (exit insert mode), **kel** (clear to end of line), **ked** (clear to end of screen), **kich1** (insert character or enter insert mode), **kil1** (insert line), **knp** (next page), **kpp** (previous page), **kind** (scroll forward/down), **kri** (scroll backward/up), **khts** (set a tab stop in this column). In addition, if the keypad has a 3 by 3 array of keys including the four arrow keys, the other five keys can be given as **ka1, ka3, kb2, kc1,** and **kc3**. These keys are useful when the effects of a 3 by 3 directional pad are needed.

Tabs and Initialization

If the terminal has hardware tabs, the command to advance to the next tab stop can be given as **ht** (usually CTRL I). A backtab command which moves leftward to the next tab stop can be given as **cbt**. By convention, if the teletype modes indicate that tabs are being expanded by the computer rather than being sent to the terminal, programs should not use **ht** or **cbt,** even if they are present, since the user may not have the tab stops properly set. If the terminal has hardware tabs that are initially set every *n* spaces when the terminal is powered up, the numeric parameter **it** is given, showing the number of spaces the tabs are set to. This is normally used by the `tset` command to determine whether to set the mode for hardware tab expansion and whether to set the tab stops. If the terminal has tab stops that can be saved in nonvolatile memory, the terminfo description can assume that they are properly set.

Other capabilities include **is1, is2,** and **is3**, initialization strings for the terminal, **iprog**, the path name of a program to be run to initialize the terminal, and **if**, the name of a file containing long initialization strings. These strings are expected to set the terminal into modes consistent with the rest of the terminfo description. They are normally sent to the terminal, by the `tset` program, each time the user logs in. They will be printed in the following order: **is1, is2**, setting tabs using **tbc** and **hts, if**, running the program **iprog**, and finally **is3**. Most initialization is done with **is2**. Special terminal modes can be set up without duplicating strings by putting the common sequences in **is2** and special cases in **is1** and **is3**. A pair of sequences that does a harder reset from a totally unknown state can be analogously given as **rs1, rs2, rf,** and **rs3**, analogous to **is2** and **if**. These strings are output by the *reset*

program, which is used when the terminal gets into a wedged state. Commands are normally placed in **rs2** and **rf** only if they produce annoying effects on the screen and are not necessary when logging in. For example, the command to set the vt100 into 80-column mode would normally be part of **is2**, but it causes an annoying movement of the screen and is not normally needed because the terminal is usually already in 80-column mode.

If there are commands to set and clear tab stops, they can be given as **tbc** (clear all tab stops) and **hts** (set a tab stop in the current column of every row). If a more complex sequence is needed to set the tabs than can be described by this, the sequence can be placed in **is2** or **if**.

Delays

Certain capabilities control padding in the teletype driver. These are primarily needed by hard copy terminals, and are used by the `tset` program to set teletype modes appropriately. Delays embedded in the capabilities **cr**, **ind**, **cub1**, **ff**, and **tab** cause the appropriate delay bits to be set in the teletype driver. If **pb** (padding baud rate) is given, these values can be ignored at baud rates below the value of **pb**.

Miscellaneous

If the terminal requires other than a null (zero) character as a pad, this can be given as **pad**. Only the first character of the **pad** string is used.

If the terminal has an extra status line that is not normally used by software, indicate this fact. If the status line is viewed as an extra line below the bottom line, into which one can cursor-address normally (such as the Heathkit h19's 25th line, or the 24th line of a vt100 which is set to a 23-line scrolling region), the capability **hs** should be given. Special strings to go to the beginning of the status line and to return from the status line can be given as **tsl** and **fsl**. The **fsl** string must leave the cursor position in the same place it was before **tsl**. If necessary, the **sc** and **rc** strings can be included in **tsl** and **fsl** to get this effect. The parameter **tsl** takes one parameter, which is the column number of the status line the cursor is to be moved to. If escape sequences and other special commands, such as tab, work while in the status line, give the **eslok** flag. A string that turns off the status line, or otherwise erases its contents, should be given as **dsl**. If the terminal has commands to save and restore the position of the cursor, give them as **sc** and **rc**. The status line is normally assumed to be the same width as the rest of the screen, for example, **cols**. If the status line is a different width (possibly because the terminal does not allow an entire line to be loaded), the width, in columns, can be indicated with the numeric parameter, **wsl**.

If the terminal can move up or down half a line, you can indicate this with **hu** (half-line up) and **hd** (half-line down). This is primarily useful for superscripts and subscripts on hardcopy terminals. If a hardcopy terminal can eject to the next page (form feed), give this as **ff** (usually CTRL L).

If there is a command to repeat a given character a given number of times (to save time transmitting a large number of identical characters), you can indicate this with the parameterized string **rep**. The first parameter is the character to be repeated and the second is the number of times to repeat it. Thus, tparm(repeat_char, 'x', 10) represents "xxxxxxxxxx".

If the terminal has a settable command character, such as the TEKTRONIX 4025, this can be indicated with **cmdch**. Choose a prototype command character to use in all capabilities. This character is given in the **cmdch** capability to identify it. The

following convention is supported on some UNIX systems: the environment is to be searched for a **CC** variable, and, if found, all occurrences of the prototype character are replaced with the character in the environment variable.

Terminal descriptions that do not represent a specific kind of known terminal, such as *switch*, *dialup*, *patch*, and *network*, should include the **gn** (generic) capability, so that programs can complain that they do not know how to talk to the terminal. This capability does not apply to *virtual* terminal descriptions, for which the escape sequences are known.

If the terminal uses xon/xoff handshaking for flow control, give **xon**. Padding information should still be included, so that routines can make better decisions about costs, but actual pad characters are not transmitted.

If the terminal has a meta key that acts as a shift key, setting the eighth bit of any character transmitted, this fact can be indicated with **km**. Otherwise, software assumes that the eighth bit is parity and it is usually cleared. If strings exist to turn this meta mode on and off, they can be given as **smm** and **rmm**.

If the terminal has more lines of memory than will fit on the screen at once, the number of lines of memory can be indicated with **lm**. A value of **lm#0** indicates that the number of lines is not fixed, but that there is still more memory than fits on the screen.

If the terminal is one of those supported by the UNIX virtual terminal protocol, the terminal number can be given as **vt**.

Media copy strings that control an auxiliary printer connected to the terminal can be given as **mc0**: print the contents of the screen, **mc4**: turn off the printer, and **mc5**: turn on the printer. When the printer is on, all text sent to the terminal is sent to the printer. It is undefined whether the text is also displayed on the terminal screen when the printer is on. A variation **mc5p** takes one parameter, and leaves the printer on for as many characters as the value of the parameter. It then turns the printer off. The parameter should not exceed 255. All text, including **mc4**, is transparently passed to the printer while an **mc5p** is in effect.

Strings to program function keys can be given as **pfkey**, **pfloc**, and **pfx**. Each of these strings takes two parameters: the function key number to program (from 0 to 10) and the string to program it with. Function key numbers out of this range may program undefined keys in a terminal-dependent manner. The difference between the capabilities is that **pfkey** causes pressing the given key to be the same as the user typing the given string; **pfloc** causes the string to be executed by the terminal in local; and **pfx** causes the string to be transmitted to the computer.

Restrictions

Hazeltine terminals, which do not allow tilde (~) characters to be displayed, should indicate **hz**.

Terminals that ignore a linefeed immediately after an **am** wrap, such as the Concept-100 and VT100, should indicate **xenl**.

If **el** is required to get rid of standout (instead of merely writing normal text on top of it), **xhp** should be given.

Teleray terminals, where tabs turn all characters moved over to blanks, should indicate **xt** (destructive tabs). This glitch is also taken to mean that it is not possible to position the cursor on top of a "magic cookie", that to erase standout mode it is instead necessary to use delete and insert line.

The Beehive Superbee, which is unable to correctly transmit the escape or CTRL C characters, has **xsb**, indicating that the f1 key is used for escape and f2 for CTRL C. (Only certain Superbees have this problem, depending on the ROM.)

Other specific terminal problems can be corrected by adding more capabilities of the form **x***x*.

Similar Terminals

If there are two very similar terminals, one can be defined as being just like the other with certain exceptions. The string capability **use** can be given with the name of the similar terminal. The capabilities given before **use** override those in the terminal type invoked by **use**. A capability can be canceled by placing **xx@** to the left of the capability definition, where xx is the capability. For example, the following entry

```
2621-nl, smkx@, rmkx@, use=2621,
```

defines a 2621-nl that does not have the **smkx** or **rmkx** capabilities, and hence does not turn on the function key labels when in visual mode. This is useful for different modes for a terminal, or for different user preferences.

Files

`/usr/lib/terminfo/?/*`
Files containing terminal descriptions

See Also

tic(1), intro(3cur), printf(3s), term(7),
Guide to X/Open curses Screen-Handling

Name

trace – system call tracer device

Description

The file `/dev/trace` is the system call trace device. It supports the following system calls: `open, close, read, ioctl,` and `select.` The device supports 16 (configurable in `sys/systrace.h` as TR_USRS) simultaneous users. It uses an 8192-byte buffer for trace records. The choice of which system calls to trace is done with the `ioctl` system call. The `select` call is used for efficient reading of the device. The `select` call uses an 8192-byte buffer and returns when the buffer is 60% full. It is required that the user use a buffer the same size as the system buffer size defined in `sys/systrace.h` as TR_BUFSIZE. All `ioctl` operations are defined in the header file, `sys/systrace.h.` The `ioctl` calls are:

ioctl	arg (pointer to)
IOTR_GETOFF	int a
IOTR_GETON	int a
IOTR_GETALL	int a
IOTR_GETPIDS	int a[10]
IOTR_GETUIDS	int a[10]
IOTR_GETSYSC	int a[10]
IOTR_GETPGRP	int a[10]
IOTR_SETOFF	int a
IOTR_SETON	int a
IOTR_SETALL	int a
IOTR_SETPIDS	int a[10]
IOTR_SETUIDS	int a[10]
IOTR_SETSYSC	int a[10]
IOTR_SETPGRP	int a[10]

Examples

A prototype example (with missing parts):

```
char cmd[BUFSIZ],buf[TR_BUFSIZ];
int pgrp[10],i;
fd = open("/dev/trace",0);      /* open the device */
pgrp[0] = dofork(cmd);          /* fork the command to trace */
for (i=1;i<TR_PGRP;i++)         /* dofork sleeps 2 seconds while */
        pgrp[i] = 0;            /* we set up to do the trace */
i = ioctl(fd,IOTR_SETPGRP,pgrp);/* set up for the trace */
/* select code goes here */
read(fd,buf,sizeof(buf));
```

See Also

trace(1), close(2), ioctl(2), open(2), read(2), select(2)

ttys(5)

Name

ttys – terminal initialization data

Description

The `/etc/ttys` file contains information used by various routines to initialize and control the use of terminal special files. This file is created when the system is installed and can be updated at any time.

Each terminal special file (`/dev/ttyxx`) has a line in the `ttys` file. Each line contains several fields. Fields are separated by tabs or spaces. A field with more than one word should be enclosed in quotes. Blank lines and comments can appear anywhere in the file. Comments begin with a number sign (#) and are terminated by a newline character. Unspecified fields default to the empty string or zero, as appropriate.

The format of each line is as follows:

```
name command type flag1 flag2 ...
```

where:

name Is the name of the special file for the terminal in the `/dev` directory. Some examples are:

```
console
ttyd0
ttyd1
```

command Is the command to be executed each time the terminal is initialized. This can happen when the system is booted, or when the superuser adds new terminals to the `ttys` file and initializes the new terminals. The command is usually `getty`, which performs such tasks as baud-rate recognition, reading the login name, and calling `login`. It can be any command you wish, such as the startup command for a window system terminal emulator or a command to maintain other daemon processes.

type Is the type of terminal normally connected to the terminal special file. You can find the possible types by examining the `/etc/termcap` file on your system. The types available are given as the third field in entries in that file. Some examples are:

```
vt100
vt200
dialup
```

flags Are the flags to be set in the `ty_status` or `ty_window` fields of the structure returned by the `getttyent`(3) routine. If the line ends in a comment, the comment is included in the `ty_comment` field of this structure.

These fields are used by the `init` command that is executed when terminals are initialized.

The `ty_status` flags are:

`on` Sets the TTY_ON bit in the `getttyent` `ty_status` field. This enables logins for this terminal.

The default if this flag is not set is that logins are disabled for the terminal.

`off` Clears the TTY_ON bit in the `getttyent` `ty_status` field. This disables logins for this terminal.

`secure` Sets the TTY_SECURE bit in the `getttyent` `ty_status` field. This allows the root user to log in on this terminal. (The `on` flag should also be set.)

The default if this flag is not set is that the root user cannot log in on this terminal.

`nomodem` Sets the TTY_LOCAL bit in the `getttyent` `ty_status` field. The line ignores modem signals. This is the default if neither the `modem` nor `nomodem` flag is set.

`modem` Clears the TTY_LOCAL bit in the `getttyent` `ty_status` field. The line recognizes modem signals.

The default if this flag is not set is `nomodem`. That is, the line does not recognize modem signals.

`shared` Sets the TTY_SHARED bit in the `getttyent` `ty_status` field. The line can be used for both incoming and outgoing connections.

The default if this flag is not set is that the line cannot be used for incoming and outgoing connections.

The `ty_window` flag is:

`window="`*`string`*`"` The quoted *string* is a window system process that `init` maintains for the terminal line.

Examples

The following example permits the root user to log in on the console at 1200 baud:

```
console "/etc/getty std.1200" vt100 on secure
```

This example allows dialup at 1200 baud without root login:

```
ttyd0 "/etc/getty d1200" dialup on
```

These two examples allow login at 9600 baud with two different terminal types: hp2621-nl and vt100. In this example, the terminals should be set up to operate in

7-bit mode, because the std.9600 gettytab entry is specified:

```
tty00 "/etc/getty std.9600" hp2621-nl on
tty01 "/etc/getty std.9600" vt100 on
```

This example shows the same two terminals as the previous example operating in full 8-bit mode. Note the use of a different gettytab entry:

```
tty00 "/etc/getty 8bit.9600" hp2621-nl on
tty01 "/etc/getty 8bit.9600" vt100 on
```

These two examples show network pseudoterminals, which should not have getty enabled:

```
ttyp0 none network
ttyp1 none network off
```

This example shows a terminal emulator and window-system startup entry and should be typed all on one line:

```
ttyv0 "/usr/bin/xterm -L -r -i -fn 9x15 =80x24+0-0 unix:0"
xterm on secure window="/usr/bin/X 0 -0 #000000 -1 #FFFFFF"
```

This example shows an example of an entry for an lta device:

```
tty01 "/etc/getty 8bit.9600" vt100 on modem secure # LAT
```

Note

Any terminal configured to run getty in 8-bit mode should specify a gettytab entry that declares 8-bit operation. The command field of the ttys entry is used to specify the gettytab entry. If the terminal device is set up to operate in 8-bit mode and the command field does not specify an 8-bit gettytab entry, output to the terminal appears as multinational characters. These characters are the result of the getty program using the eighth bit of each character to represent parity attributes. By using an 8-bit gettytab entry, the high order bit of each character is unaffected by the getty program. The examples presented demonstrate the use of both 7- and 8-bit terminals.

Files

/etc/ttys	The full pathname for the file

See Also

login(1), getttyent(3), gettytab(5), getty(8), init(i)
Guide to System Environment Setup

Name

types – primitive system data types

Syntax

#include <sys/types.h>

Description

The data `types` defined in the include file are used in UNIX system code. Some data of these `types` are accessible to user code:

```
#ifndef _TYPES_
#define _TYPES_

/* major part of a device */
#define major(x)        ((int)(((unsigned)(x)>>8)&0377))

/* minor part of a device */
#define minor(x)        ((int)((x)&0377))

/* make a device number */
#define makedev(x,y)    ((dev_t)(((x)<<8) | (y)))

typedef unsigned char   u_char;
typedef unsigned short  u_short;
typedef unsigned int    u_int;
typedef unsigned long   u_long;
typedef unsigned short  ushort;         /* sys III compat */

#ifdef mips
typedef struct  _physadr { int r[1]; } *physadr;
typedef struct  label_t {
        int     val[12];
} label_t;
#endif
typedef struct  _quad { long val[2]; } quad;
typedef long    daddr_t;
typedef char *  caddr_t;
typedef u_long  ino_t;
typedef long    swblk_t;
typedef int     size_t;
typedef int     time_t;
typedef short   dev_t;
typedef int     off_t;

typedef struct  fd_set { int fds_bits[1]; } fd_set;
#endif
```

The form *daddr_t* is used for disk addresses except in an i-node on disk. For further information, see `fs`(5). 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 and are installation-dependent. 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

dbx(1), lseek(2), time(3), fs(5)

Name

tzfile – time zone information

Syntax

#include <tzfile.h>

Description

The time zone information files used by `tzset` begin with bytes reserved for future use, followed by three 4-byte values of type "long", written in a "standard" byte order (the high-order byte of the value is written first). These values are, in order:

tzh_timecnt The number of transition times for which data is stored in the file.

tzh_typecnt The number of local time types for which data is stored in the file (must not be zero).

tzh_charcnt The number of characters of "time zone abbreviation strings" stored in the file.

This header is followed by `tzh_timecnt` 4-byte values of type "long", sorted in ascending order. These values are written in "standard" byte order. Each is used as a transition time (as returned by `time` at which the rules for computing local time change). Next come `tzh_timecnt` 1-byte values of type "unsigned char". Each one tells which of the different types of local time types described in the file is associated with the same-indexed transition time. These values serve as indices into an array of *ttinfo* structures that appears next in the file; these structures are defined as follows:

```
struct ttinfo {
        long            tt_gmtoff;
        int             tt_isdst;
        unsigned int    tt_abbrind;
};
```

Each structure is written as a 4-byte value for `tt_gmtoff` of type "long", in a standard byte order, followed by a 1-byte value for `tt_isdst` and a 1-byte value for `tt_abbrind`. In each structure, `tt_gmtoff` gives the number of seconds to be added to GMT, `tt_isdst` tells whether `tm_isdst` should be set by `localtime` and `tt_abbrind` serves as an index into the array of time zone abbreviation characters that follows the `ttinfo` structure or structures in the file.

The `localtime` call uses the first standard-time `ttinfo` structure in the file (or simply the first `ttinfo` structure, in the absence of a standard-time structure) if either `tzh_timecnt` is zero or the time argument is less than the first transition time recorded in the file.

See Also

ctime(3)

Name

ufs – ULTRIX local file system

Description

The ULTRIX file system is a local file system implemented under the Generic File System Interface, GFSI, which is described in `gfsi`(5). UFS is a reorganization of the file system that is always supplied with ULTRIX.

Aside from the difference in mounting and unmounting file systems, there are no observable differences in the file system compared to earlier releases. ULTRIX file system disks are completely transportable between releases of ULTRIX.

See Also

getmnt(2), getdirentries(2), mount(2), fstab(5), nfs(5nfs), fsck(8), mount(8)

USERFILE(5)

Name

USERFILE – defines uucp security

Syntax

/usr/lib/uucp/USERFILE

Description

The `uucp` utility uses the USERFILE to establish what access a remote system can have to the local system. An entry should exist for each system. If no entries exist for a particular system, the default entries are used. The entries for particular systems have the following format:

login-name , *node-name* **X** # *path-name*

login-name	The name with which the remote system logs in.
node-name	The name of the remote node.
X#	The execution level for the remote system. The remote system can execute commands defined in the `L.cmds`(5) file that have an execution level less than or equal to the number #.
path-name	The remote system can access anything at the local system with this prefix.

Two entries must also be provided for systems not otherwise listed:

remote, X# *path-name* ...
local, X# *path-name* ...

These entries define the execution level and access pathnames for the local system and all remote systems not defined by specific entries.

Examples

```
remote, X1 /usr/spool/uucppublic
local, X9 /
max,systemY /usr/sources /usr/src/share
max,systemZ X3 /usr
```

In the above example, the node named systemY with the login name max has access to anything with the pathname prefixes `/usr/sources` and `/usr/src/share`. The node named systemZ with the login name max can execute commands defined in `L.cmds`(5) with an execution level of 3 or lower. It can access anything with the pathname prefix `/usr`.

Any other remote systems can execute commands defined in `L.cmds`(5) with an execution level of 1 or 0. They can access anything with the pathname prefix of `/usr/spool/uucppublic`.

Users on the local system can execute any of the commands defined in `L.cmd` and access anything on the system.

See Also

Guide to the uucp Utility

utmp(5)

Name

utmp, wtmp – login records

Syntax

#include <utmp.h>

Description

The utmp file records information about who is currently using the system. The file is a sequence of entries with the following structure declared in the include file:

```
struct utmp {
        char    ut_line[8];             /* tty name */
        char    ut_name[8];             /* user id */
        char    ut_host[16];            /* host name, if remote */
        long    ut_time;                /* time on */
};
```

This structure gives the name of the special file associated with the user's terminal, the user's login name, and the time of the login in the form of time(3c).

The wtmp file records all logins and logouts. A null user name indicates a logout on the associated terminal. A terminal referenced with a tilde (~) indicates that the system was rebooted at the indicated time. The adjacent pair of entries with terminal names referenced by a vertical bar (|) or a right brace (}) indicate the system-maintained time just before and just after a date command has changed the system's timeframe.

The wtmp file is maintained by login(1) and init(8). Neither of these programs creates the file, so, if it is removed, record-keeping is turned off. It is summarized by ac(8).

Files

```
/etc/utmp
/usr/adm/wtmp
```

See Also

last(1), lastcomm(1), login(1), who(1), ac(8), init(8)

Name

uuencode – format of an encoded uuencode file

Description

Files output by `uuencode`(1c) consist of a header line, followed by a number of body lines, and a trailer line. The `uudecode` command ignores any lines preceding the header or following the trailer. Lines preceding a header must not, of course, look like a header.

The header line is distinguished by having the first six characters by the word "begin", followed by a space. The next item on the line is a mode (in octal) and a string which names the remote file. A space separates the three items in the header line.

The body consists of a number of lines, each at most 62 characters long including the trailing new line. These consist of a character count, followed by encoded characters, followed by a new line. The character count is a single printing character and represents an integer, the number of bytes the rest of the line represents. Such integers are always in the range from 0 to 63 and can be determined by subtracting the character space (octal 40) from the character.

Groups of 3 bytes are stored in 4 characters, with 6 bits per character. All are offset by a space to make the characters print. The last line may be shorter than the normal 45 bytes. If the size is not a multiple of 3, this fact can be determined by the value of the count on the last line. Extra dummy characters are included to make the character count a multiple of 4. The body is terminated by a line with a count of zero. This line consists of one ASCII space.

The trailer line consists of "end" on a line by itself.

See Also

mail(1), uucp(1c), uudecode(1c), uuencode(1c), uusend(1c)

Name

varargs – handle variable argument list

Syntax

#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 that 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*. A semicolon should not 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 Returns 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. This information cannot be determined at run time.

va_end is used to clean up.

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

The calling routine must specify how many arguments there are, because 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. The `printf` routine can tell how many arguments there are by the format.

It is nonportable to specify a second argument of *char*, *short*, or *float* to *va_arg*, because 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.

Examples

The following example presents an 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), vprintf(3s)

ypfiles (5yp)

Name

ypfiles – Yellow Pages data base and directory structure

Description

The Yellow Pages (YP) data base lookup service uses a data base of dbm files in the /etc/yp directory hierarchy. A dbm data base consists of two files, created by calls to the dbm(3x) library package. One has the filename extension .pag and the other has the filename extension .dir. For instance, the data base named hosts.byname, is implemented by the pair of files hosts.byname.pag and hosts.byname.dir.

A dbm data base served by the YP is called a YP *map*. A YP *domain* is a named set of YP maps. Each YP domain is implemented as a subdirectory of /etc/yp containing the map. The number of YP domains that can exist is unlimited. Each domain can contain any number of maps.

The YP maps are not required by the YP lookup service, although they may be required for the normal operation of other parts of the system. The YP lookup service serves all maps. If the map exists in a given domain and a client asks about it, the YP will serve it. There is, however, a set of default maps that the YP service serves. The files representing these maps are listed in this description under Files.

For a map to be accessible consistently, it must exist on all YP servers that serve the domain. To provide data consistency between the replicated maps, an entry to execute the ypxfr command periodically should be made in the /usr/lib/crontab file on each slave server. More information on this topic is in ypxfr(8yp). An entry in the /etc/lib/crontab file must not exist, either on a YP master server or on a pure YP client machine.

The YP maps should contain two distinguished key-value pairs. The first is the key YP_LAST_MODIFIED, whose value is a 10-character ASCII order number. The order number should be the UNIX time in seconds when the map was built. The second key is YP_MASTER_NAME, with the name of the YP master server as a value. The makedbm(8yp) command generates both key-value pairs automatically. A map that does not contain both key-value pairs can be served by the YP, but the ypserv process will not be able to return values for "Get order numbe"r or "Get master name" requests. In addition, values of these two keys are used by ypxfr(8yp) when it transfers a map from a master YP server to a slave.

Before they can be properly accessed, the YP maps must be initially set up for both masters and slaves by using the ypsetup(8yp) function. Further, YP maps must be generated and modified only at the master server location. Copies of the master server YP maps can then be transferred to the slave servers using the ypxfr(8yp) function. If ypxfr is unable to determine a map's location, or if it is unable to determine whether the local copy is more recent than the master copy, extra command line switches must be set when it is executed.

After the server data bases are set up, the contents of some maps may change. In general, some ASCII source version of the data base exists on the master. This version should be changed with a standard text editor. The update is incorporated into the YP map and is propagated from the master to the slaves by running /etc/yp/Makefile. All maps must have entries in /etc/yp/Makefile. If a YP map is added, the /etc/yp/Makefile must be edited to support the new

map. The makefile uses makedbm(8yp) to generate the YP map on the master, and yppush(8yp) to propagate the changed map to the slaves. The yppush(8yp) command is a client of the map *ypservers*, which lists all of the YP servers.

Files

```
/etc/passwd
/etc/group
/etc/hosts
/etc/networks
/etc/services
/etc/protocols
/etc/netgroup
/etc/ethers
```

See Also

makedbm(8yp), rpcinfo(8nfs), ypmake(8yp), yppoll(8yp), yppush(8yp), ypserv(8yp), ypsetup(8yp), ypxfr(8yp)

Index

E

F

G

H

I

K

L

M

N

P

R

S

T

U

W

Y

How to Order Additional Documentation

Technical Support

If you need help deciding which documentation best meets your needs, call 800-343-4040 before placing your electronic, telephone, or direct mail order.

Electronic Orders

To place an order at the Electronic Store, dial 800-234-1998 using a 1200- or 2400-baud modem from anywhere in the USA, Canada, or Puerto Rico. If you need assistance using the Electronic Store, call 800-DIGITAL (800-344-4825).

Telephone and Direct Mail Orders

Your Location	Call	Contact
Continental USA, Alaska, or Hawaii	800-DIGITAL	Digital Equipment Corporation P.O. Box CS2008 Nashua, New Hampshire 03061
Puerto Rico	809-754-7575	Local Digital Subsidiary
Canada	800-267-6215	Digital Equipment of Canada Attn: DECdirect Operations KAO2/2 P.O. Box 13000 100 Herzberg Road Kanata, Ontario, Canada K2K 2A6
International	————	Local Digital subsidiary or approved distributor
Internal*	————	SSB Order Processing - WMO/E15 *or* Software Supply Business Digital Equipment Corporation Westminster, Massachusetts 01473

* For internal orders, you must submit an Internal Software Order Form (EN-01740-07).

Reader's Comments

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

Please rate this manual:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

What would you like to see more/less of? _____

What do you like best about this manual? _____

What do you like least about this manual? _____

Please list errors you have found in this manual:

Page	Description
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____

Additional comments or suggestions to improve this manual:

What version of the software described by this manual are you using? _____

Name/Title _____ Dept. _____

Company _____ Date _____

Mailing Address _____

_____ Email _____ Phone _____

Do Not Tear – Fold Here and Tape

digital™

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST–CLASS MAIL PERMIT NO. 33 MAYNARD MA

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
OPEN SOFTWARE PUBLICATIONS MANAGER
ZKO3–2/Z04
110 SPIT BROOK ROAD
NASHUA NH 03062–9987

Do Not Tear – Fold Here

Cut
Along
Dotted
Line