IRIX Programmer's Reference Manual

Volume III

Section 4
Section 5
Permuted Index

IRIS-4D Series

SiliconGraphics
Computer Systems

Document number: 007-0602-050

IRIX Programmer's Reference Manual

Volume III

Version 5.0

Document Number 007-0602-050

IRIX Programmer's Reference Manual
Version 5.0
Document Number 007-0602-050

Silicon Graphics, Inc.
Mountain View, California

TABLE OF CONTENTS

4. File Formats

5. Miscellaneous Facilities

NAME

intro – introduction to file formats

DESCRIPTION

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

NAME

a.out – assembler and link editor output

SYNOPSIS

#include <a.out.h>

DESCRIPTION

a.out is the output file format 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. The debugger uses the *a.out* file to provide symbolic information to the user.

IRIX and the MIPS compilers use a file format that is similar to standard AT&T System V COFF (common object file format). For more information, see the *Assembly Language Programmer's Guide*.

The MIPS File Header definition is based on the System V header file *filehdr.h* with the following changes (also see *filehdr*(4)):

- The symbol table file pointer, *f_symptr*, and the number of symbol table entries *f_nsyms*, now 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 has 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*(4)).

The MIPS relocation information definition is similar to that in Berkeley 4.3 UNIX, which has "local" relocation types (see *reloc*(4)). Also see the section entitled **Section Relocation Information** in the *Assembly Language Programmer's Guide* for more detailed information.

For more information about System V COFF, refer to the AT&T UNIX System V Support Tools Guide.

The MIPS file format follows this scheme:

- File Header
- Optional Headers
- Section Headers
- Section Data—includes text, read-only data, large data, 8 and 4-byte literal pools, small data, small bss (0 size), large bss (0 size), and shared library information.
- Section Relocation Information—includes text, read-only data, large data, 8 and 4-byte literal pools, and small data.
- Global Pointer (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 non-globals or fully stripped.
- Procedure Descriptor Table—missing if fully stripped.
- Local Symbols—missing if stripped of non-globals or if fully stripped.
- Optimization Symbols—created only if debugging is on, and missing if stripped of non-globals or fully stripped.
- Auxiliary Symbols—created only if debugging is on, and missing if stripped of non-globals or fully stripped.
- Local Strings—missing if stripped of non-globals or if fully stripped.
- External Strings—missing if fully stripped.
- Relative File Descriptor—missing if stripped of non-globals or if fully stripped.
- File Descriptors—missing if stripped of non-globals or if fully stripped.
- External Symbols—missing is fully stripped.

The Section Data

MIPS files are represented in several sections: *.text*, *.rdata* (read-only data), *.data* (data), *.sdata* (small data), *.lit8* (8-byte literal pool), *.lit4* (4-byte literal pool), *.sbss* (small block started by storage), *.bss* (block started by storage), *.init* (initialization), and *.lib* (shared library references).

Generally only sections actually needed in an *a.out* file are present in the file.

The *.text* section contains the machine instructions that are to be executed; the *.rdata*, *.data*, *.sdata*, *.lit8*, and *.lit4* sections contain initialized data; the *.sbss* and *.bss* sections reserve space for ininitialized data that is created by the kernel loader for the program before execution and filled with zeros.

The *.init* section contains shared library interface initialization information. The *.lib* section contains references to the shared libraries this *a.out* file uses.

SEE ALSO

Assembly Language Programmer's Guide
as(1), ld(1), nm(1), dbx(1), strip(1), filehdr(4), scnhdr(4), reloc(4), syms(4), linenum(4)

NAME

acct – per-process accounting file format

SYNOPSIS

#include <sys/acct.h>

DESCRIPTION

Files produced as a result of calling *acct*(2) have records in the form defined by *<sys/acct.h>*, whose contents are:

```
typedef ushort comp_t;      /* "floating point" */
                    /* 13-bit fraction, 3-bit exponent  */

struct   acct
{
        char    ac_flag;        /* Accounting flag */
        char    ac_stat;        /* Exit status */
        ushort  ac_uid;         /* Accounting user ID */
        ushort  ac_gid;         /* Accounting group ID */
        dev_t   ac_tty;         /* control typewriter */
        time_t  ac_btime;       /* Beginning time */
        comp_t  ac_utime;       /* acctng user time in clock ticks */
        comp_t  ac_stime;       /* acctng system time in clock ticks */
        comp_t  ac_etime;       /* acctng elapsed time in clock ticks */
        comp_t  ac_mem;         /* memory usage in clicks */
        comp_t  ac_io;          /* chars trnsfrd by read/write */
        comp_t  ac_rw;          /* number of block reads/writes */
        char    ac_comm[8]; /* command name */
};

extern  struct  acct            acctbuf;
extern  struct  inode           *acctp; /* inode of accounting file */

#define AFORK 01                /* has executed fork, but no exec */
#define ASU   02                /* used super-user privileges */
#define ACCTF 0300              /* record type: 00 = acct */
```

In *ac_flag*, the AFORK flag is turned on by each *fork*(2) and turned off by an *exec*(2). The *ac_comm* field is inherited from the parent process and is reset by any *exec*. Each time the system charges the process with a clock tick, it also adds to *ac_mem* the current process size, computed as follows:

(data size) + (text size) / (number of in-core processes using text)

The value of *ac_mem / (ac_stime + ac_utime)* can be viewed as an approximation to the mean process size, as modified by text sharing.

The structure acct, which resides with the source files of the accounting commands, represents the total accounting format used by the various accounting commands:

```
/*
 * total accounting (for acct period), also for day
 */

struct tacct {
        uid_t           ta_uid;     /* userid */
        char            ta_name[8]; /* login name */
        float           ta_cpu[2];  /* cum. cpu time, p/np (mins) */
        float           ta_kcore[2]; /* cum kcore-minutes, p/np */
        float           ta_con[2];  /* cum. connect time, p/np, mins */
        float           ta_du;      /* cum. disk usage */
        long            ta_pc;      /* count of processes */
        unsigned short  ta_sc;      /* count of login sessions */
        unsigned short  ta_dc;      /* count of disk samples */
        unsigned short  ta_fee;     /* fee for special services */
};
```

SEE ALSO

acct(2), exec(2), fork(2) in the *Programmer's Reference Manual.*
acct(1M) in the *System Administrator's Reference Manual.*
acctcom(1) in the *User's Reference Manual.*

BUGS

The *ac_mem* value for a short-lived command gives little information about the actual size of the command, because *ac_mem* may be incremented while a different command (e.g., the shell) is being executed by the process.

NAME

aliases – aliases file for sendmail

SYNOPSIS

/usr/lib/aliases

DESCRIPTION

This file describes user id aliases used by */usr/lib/sendmail*. It is formatted as a series of lines of the form

name: name_1, name2, name_3, . . .

The *name* is the name to alias, and the *name_n* are the aliases for that name. Lines beginning with white space are continuation lines. Lines beginning with '#' are comments.

Aliasing occurs only on local names. Loops can not occur, since no message will be sent to any person more than once.

After aliasing has been done, local and valid recipients who have a ".forward" file in their home directory have messages forwarded to the list of users defined in that file.

This is only the raw data file; the actual aliasing information is placed into a binary format in the files */usr/lib/aliases.dir* and */usr/lib/aliases.pag* using the program *newaliases*(1). A *newaliases* command should be executed each time the aliases file is changed for the change to take effect.

SEE ALSO

newaliases(1), sendmail(1M)
SENDMAIL Installation and Operation Guide.
SENDMAIL: An Internetwork Mail Router.

BUGS

Because of restrictions in *dbm*(3B) a single alias cannot contain more than about 1000 bytes of information. You can get longer aliases by "chaining"; that is, make the last name in the alias be a dummy name which is a continuation alias.

NAME

ar – archive (library) file format

SYNOPSIS

#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];
};
typedef struct ar_hdr ARHDR;
```

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.

SEE ALSO

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

BUGS

File names lose trailing blanks. Most software dealing with archives takes even an included blank as a name terminator.

NAME

autologin – set autologin user identity

SYNOPSIS

/etc/autologin

DESCRIPTION

/etc/autologin is an ASCII file containing the login user name to be used when autologin is enabled. The file is used by **getty(1M)** and **login(1)** when automatically initiating a terminal session on the graphics console.

FILES

/etc/autologin

SEE ALSO

getty(1M),
login(1) in the *User's Reference Manual.*

NAME

cftime – language specific strings

DESCRIPTION

The programmer can create one printable file per language. These files must be kept in a special directory /lib/cftime. If this directory does not exist, the programmer should create it. The contents of these files are:

- abbreviated month names (in order)
- month names (in order)
- abbreviated weekday names (in order)
- weekday names (in order)
- default strings that specify formats for local time (%x) and local date (%X).
- default format for cftime, if the argument for cftime is zero or null.
- AM (ante meridian) string
- PM (post meridian) string

Each string is on a line by itself. All white space is significant. The order of the strings in the above list is the same order in which the strings appear in the file shown below.

EXAMPLE

/lib/cftime/usa_english

```
Jan
Feb
 ...
January
February
 ...
Sun
Mon
 ...
Sunday
Monday
 ...
%H:%M:%S
%m/%d/%y
%a %b %d %T %Z %Y
AM
```

PM

FILES

/lib/cftime – directory that contains the language specific printable files (create it if it does not exist)

SEE ALSO

ctime(3C) in the *Programmer's Reference Manual.*

NAME

core – format of core image file

SYNOPSIS

#include <core.out.h>

DESCRIPTION

The IRIX system writes out a core image of a terminated process when any of various errors occur. See *signal*(2) for the list of reasons; the most common are memory violations, illegal instructions, bus errors, and user-generated quit signals. The core image is called *core* and is written in the process's working directory (provided it can be; normal access controls apply). A process with an effective user ID different from the real user ID will not produce a core image.

The format of the core image is defined by **<core.out.h>.** It consists of a header, maps, descriptors, and section-data.

The header data includes the process name (as in *ps*(1)), the signal that caused the core-dump, the descriptor array, and the corefile location of the map array.

Each descriptor defines the length of useful process data. One descriptor defines the general-purpose registers at the time of the core-dump for example. The data is present in the core image at the file-location given in the descriptor only if the **IVALID** flag is set in the descriptor.

Each map defines the virtual address and length of a section-of-the-process at the time of the core-dump. The data is present in the core image at the file-location given in the descriptor only if the **VDUMPED** flag is set in the map. The process' stack, and data sections are normally written in the core image. The process' text is not normally written in the core image.

NOTE

Core image format designed by Silicon Graphics, Inc.

SEE ALSO

edge(1), dbx(1), ps(1), setuid(2), signal(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%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);
```

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

stat(2).

cpio(1), find(1) in the *User's Reference Manual.*

NAME

cshrc – system-wide csh initialization command file

DESCRIPTION

The file **/etc/cshrc** contains a list of commands to be invoked whenever a user logs into the system with *csh*(1) as their login shell. These commands are executed before those in the **.cshrc** and **.login** files in the home directory of the user.

FILES

/etc/cshrc

SEE ALSO

csh(1).

NAME

dbg, debug – the debug file system

SYNOPSIS

#include <sys/types.h>
#include <sys/fs/dbfcntl.h>

DESCRIPTION

The debug file system, normally mounted under /debug, provides an interface to running processes that may be used by debuggers such as *dbx*. The "files" of this file system are of the form */debug/<pid>*, where *<pid>* is the process id of a running process. These files actually consume no disk space, and are only convenient handles by which a debugger can attach to a process. The debugger does so by opening the desired /debug file with the *open(2)* system call. The debugger may perform various *fcntl(2)* commands to the process; for example, to suspend and restart the process. When the process is suspended, ordinary *seek(2)*, *read(2)*, and *write(2)* system calls will access the process' address space.

The *statfs(2)* system call will return valid information concerning the *dbg* file system. The total and free blocks as reported by *df(1)* respectively represent the total virtual memory (real memory plus swap space) available and currently free.

The following *fcntl(2)* codes are recognized by *dbg* files. An optional argument may be supplied to the call; the form of the argument varies with the request and will be described where appropriate.

DFCSTOP

Suspend the process as soon as possible. This usually leaves the process in the stopped state; this can be verified with **ps(1)**. However, sleeping processes remain on the sleep queue and are marked as suspended, and are therefore can be considered stopped.

DFCWSTOP

Wait for the process to stop. This is useful after setting breakpoints, system call trace masks, etc.

DFCRUN

Resume process execution after process is suspended. The argument specifies how pending signals are to be treated. If this is CLEARNOSIG, no signals are cleared; if this is CLEARCURSIG, only the current signal is cleared; or if this is CLEARALLSIG, all pending signals are cleared. If the value is between 1 and NSIG inclusive, clear all pending signals and continue with the given signal.

DFCSSTEP
Resume process execution, but stop after executing a single machine instruction. The argument is the same as that of the DFCRUN request.

DFCCSIG
Clear the highest signal pending. This request takes no arguments. (This request is really not too useful, since the argument to DFCRUN or DFCSSTEP provides for more precise signal control.)

DFCKILL
Send a signal to the process. The argument is a an unsigned int representing the signal number.

DFCSMASK
Sets the signal trace mask. The argument is a pointer to a long indicating whether the process should stop upon receiving specified signals. To trace signal *s*, set *(1<<(s-1))* in the signal mask.

DFCGMASK
Retrieve the signal trace mask. The argument is a pointer to a long in which to store the current signal mask.

DFCSENTRYMASK
Sets the system call entry mask. The argument is a pointer to an array of SYSMASKLEN long values indicating whether the process should stop prior to executing certain system calls. For example, to trace system call *syscallno* the code would resemble something like:

```
#include <sys/user.h>                /* defines SYSMASKLEN */
#include <sys/fs/dbfcntl.h>
#define  BITSPERLONG 32
long entrymask[SYSMASKLEN]
        .
        .
        .
entrymask[syscallno/BITSPERLONG] |=
        (1 << (syscallno%BITSPERLONG));
fcntl(fd, DFCSENTRYMASK, &entrymask);
```

Note that the system call numbers as defined in */usr/include/sys.s* are relative to a large offset, *SYSVoffset*. This offset should be subtracted from the actual system call number before setting the bit in the mask.

DFCGENTRYMASK

Sets the system call exit mask. As above, the argument is a pointer to an array of SYSMASKLEN long values; however, this indicates that the process should stop immediately after executing specified system calls.

DFCABORT

Abort the system call in progress. If the process is stopped prior to executing a system call (see DFCSENTRYMASK above), this request will cause the signal call to return prematurely with an EINTR error code when the process is resumed.

DFCSEXEC

Have the process stop after an *exec(2)* system call. This is useful for stopping the process to set initial breakpoints.

DFCREXEC

No longer stop the process after an *exec(2)* system call.

DFCGETREGS

Retrieve the processor registers and signal handlers. The argument should be a pointer to an array of NPTRC_REGS unsigned integers. The general purpose processor registers, floating point registers, signal handlers, and special purpose control registers will be deposited in this array. NPTRC_REGS, as well as a layout of the registers in the array, is defined in the header file */usr/include/sys/ptrace.h*.

DFCPUTREGS

Write the processor registers. The argument should be a pointer to an array of NPTRC_REGS unsigned integers. The contents of the array are copied to the appropriate register locations. Note that the signal handlers and read-only control registers are not modified.

DFCGETPRINFO

Retrieve some useful information about the process. The argument to this request should be a pointer to a buffer of **sizeof(struct procinfo)** bytes. This structure, defined in */usr/include/sys/fs/dbfcntl.h*, contains useful information concerning the process status.

DFCGETPR

Get the entire contents of the process table entry corresponding to the process. The argument supplied should be a pointer to a buffer of at least **sizeof(struct proc)** bytes. (Due to the lack of portability of this call, the DFCGETPRINFO request is preferred.)

DFCOPENT

If the given argument is zero, return the file descriptor corresponding to the text region of the process. If the given argument is non-zero, interpret this value as a virtual address of the process. Return the file descriptor corresponding to the region containing this address. This call may be used to locate symbol tables of the process.

DFCEXCLU

If the given argument is zero, make the text region "private" and writable. Otherwise, make the region corresponding to the given address private. (Note: the first write to a read-only region will have the same affect and thus this call is usually not necessary.)

DIAGNOSTICS

These return codes are relevant to most system calls applied to *dbg* files.

[ENOENT]	The process no longer exists or is in the zombie state.
[EACCES]	The caller does not have permission to trace the process.
[EIO]	The process should be stopped before attempting this call.
[EINVAL]	An argument to this call was invalid.
[EFAULT]	An address supplied by the user was invalid.
[EPERM]	This operation is not allowed on a *dbg* file.
[EROFS]	The *dbg* filesystem was mounted read only.
[ENOMEM]	The kernel could not allocate enough memory to perform this request.

SEE ALSO

dbx(1), fcntl(2)

NAME

dir – format of EFS directories

SYNOPSIS

#include <sys/fs/efs_dir.h>

DESCRIPTION

A directory behaves exactly like an ordinary file, save 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 [see *inode*(4)]. The EFS directory format supports variable length names of up to 255 characters.

DIRECTORY BLOCKS

Each EFS directory is segmented into directory blocks defined by the following data structure:

```
#define EFS_DIRBLK_HEADERSIZE 4
struct    efs_dirblk {
          /* begin header */
          ushort  magic;
          unchar firstused;
          unchar slots;
          /* end header */

          /* rest is space for efs_dent's */
          unchar space[EFS_DIRBSIZE - EFS_DIRBLK_HEADERSIZE];
};
```

Each directory block is subdivided into three separate areas: a header, an array of entry offsets and an array of directory entries. The system restricts directory entries to short boundaries and stores offsets in the directory block compacted by shifting them right by one.

The header area contains a *magic* number to identify the block as being a directory block. If the *magic* number is incorrect, the operating system will refuse to manipulate the directory, thus avoiding further corruption.

The array of entry offsets immediately follows the header and is sized according to the directories contents and contains compacted offsets which point to each directory entry. The number of entry offsets available is kept in *slots*. The *firstused* field contains a compacted offset which positions the first byte of the directory entries.

The space between the end of the entry array and the beginning of the directory entries (*firstused*) is free space which the system uses for allocating new directory entries and entry offsets. The system keeps the free space in a directory block compacted by coalescing holes created by entry removal. When a directory entry is removed, the system adjusts the entry offsets

for all entries that move. Also, the entry offset for the removed entry is zeroed. If the removed entry was the last in the entry offset array, the number of *slots* is reduced. Directory entries never change which entry offset they use.

DIRECTORY ENTRIES

Directory entries have the following structure:

```
struct    efs_dent {
          union {
                ulong  l;
                ushort s[2];
          } ud_inum;
          unchar d_namelen;
          char   d_name[3];
};
```

The *d_name* field is actually of variable size, depending upon the value contained in *d_namelen*. The system pads out the directory entry to insure that it begins on a short boundary in the directory block. The *ud_inum* field contains the entries inode number.

SEE ALSO

fs(4), inode(4).

NAME

dirent – file system independent directory entry

SYNOPSIS

#include <sys/dirent.h>

DESCRIPTION

Different file system types may have different directory entries. The *dirent* structure defines a file system independent directory entry, which contains information common to directory entries in different file system types. A set of these structures is returned by the *getdents*(2) system call.

The *dirent* structure is defined below.

```
struct dirent {
        long            d_ino;
        off_t           d_off;
        unsigned short  d_reclen;
        char            d_name[1];
};
```

The **d_ino** is a number which is unique for each file in the file system. The field **d_off** is an opaque offset (i.e., not necessarily in bytes) of the next directory entry in the actual file system directory. The field **d_name** is the beginning of the character array giving the name of the directory entry. This name is null terminated and may have at most **MAXNAMLEN** characters. This results in file system independent directory entries being variable length entities. The value of **d_reclen** is the record length of this entry. This length is defined to be the number of bytes between the current entry and the next one, so that it will always result in the next entry being on a long boundary.

SEE ALSO

directory(3X), getdents(2).

NAME

filehdr – file header for MIPS object files

SYNOPSIS

#include <filehdr.h>

DESCRIPTION

Every MIPS 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 */
};
```

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

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

MIPS 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 MIPS 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(4).

NAME

efs – layout of the Extent file system

SYNOPSIS

#include <sys/param.h>
#include <sys/fs/efs.h>

DESCRIPTION

Every Extent file system storage volume has a common format for certain vital information. Every such volume is divided into a certain number of 512 byte long sectors, also called *basic blocks*. Basic block 0 is unused and is available to contain a bootstrap program or other information.

Basic block 1 is the *super-block*. The format of an Extent file system super-block is:

```
/*
 * Structure of the super-block for the Extent file system
 */
struct    efs {
    /*
     * This portion is read off the volume
     */
    long  fs_size;          /* size of file system, in sectors */
    long  fs_firstcg;       /* bb offset to first cg */
    long  fs_cgfsize;       /* size of cylinder group in bb's */
    short fs_cgisize;       /* bb's in inodes per cylinder group */
    short fs_sectors;       /* sectors per track */
    short fs_heads; /* heads per cylinder */
    short fs_ncg;           /* # of groups in file system */
    short fs_dirty;         /* fs needs to be fsck'd */
    time_t          fs_time;            /* last super-block update */
    long  fs_magic;/* magic number */
    char  fs_fname[6];      /* file system name */
    char  fs_fpack[6];      /* file system pack name */
    long  fs_bmsize;        /* size of bitmap in bytes */
    long  fs_tfree;         /* total free data blocks */
    long  fs_tinode;        /* total free inodes */
    long  fs_bmblock;       /* bitmap location */
    long  fs_replsb;/* location of replicated superblock. */
    char  fs_spare[24];     /* space for expansion */
    long  fs_checksum;      /* checksum of volume portion of fs */
    /*
     * The remainder of this structure, defined fully in <sys/fs/efs_sb.h>
     * is used by the operating system only.
     */
```

```
};

#define EFS_MAGIC     0x072959
```

Note that the struct efs that is defined in *<sys/fs/efs_sb.h>* contains more fields. The extra fields are used internally by the operating system, and are not discussed here. *fs_size* holds the size in basic blocks of the file system. This variable is filled in when the file system is first created with *mkfs*(1M).

fs_firstcg contains the basic block offset to the first *cylinder group*. There are *fs_ncg* cylinder groups contained in the file system. Each cylinder group is composed of *fs_cgfsize* basic blocks, of which *fs_cgisize* basic blocks are used for inodes.

fs_sectors, and *fs_heads* are used to specify the geometry of the underlying disk containing the file system. Note that *fs_heads* is in fact currently unused, and should not be relied upon.

fs_dirty is a flag which indicates if the file system needs to be checked by the *fsck*(1M) program. The *fs_time* field contains the time stamp of when the file system was last modified. *fs_name* holds the *name* of the file system (where it is mounted, more or less) while *fs_fpack* contains which volume this file system is. The *fs_fpack* field is singularly useless, but is provided for utility compatibility. *fs_magic* is used to tag the superblock of the file system as an Extent file system.

The *fs_bmsize* field contains, in bytes, the size of the data block bitmap. The data block bitmap is used for data block allocation. Each one in the bitmap indicates a free block. The *fs_bmblock* field contains the location of the bitmap if it has been moved from its default location (basic block 2) because the file system has been constructed on a logical volume which has been extended (see *growfs(1m)*).

fs_tfree and *fs_tinode* contain the total free blocks and inodes, respectively.
The *fs_replsb* field contains the location of a replicated superblock, if one exists.
The *fs_spare* field is reserved for future use.
Lastly, the *fs_checksum* variable holds a checksum of the above fields (not including itself).

During the *mount*(1M) of the file system, the *fs_dirty* and *fs_checksum* fields are examined. If *fs_dirty* is non-zero, or the *fs_checksum* variable does not match the systems computed checksum, then the file system must be cleaned with *fsck* before it can be mounted. If the file system is the *root* partition, then this check is ignored, as it is necessary to be able to run *fsck* on a dirty *root* from a dirty *root*. For the format of an inode and its flags, see *inode*(4).

FILES

/usr/include/sys/fs/efs*.h
/usr/include/sys/stat.h

SEE ALSO

fsck(1M), mkfs(1M), inode(4).

NAME

fspec – format specification in text files

DESCRIPTION

It is sometimes convenient to maintain text files on the UNIX system with non-standard tabs, (i.e., tabs which are not set at every eighth column). Such files must generally be converted to a standard format, frequently by replacing all tabs with the appropriate number of spaces, before they can be processed by UNIX system commands. A format specification occurring in the first line of a text file specifies how tabs are to be expanded in the remainder of the file.

A format specification consists of a sequence of parameters separated by blanks and surrounded by the brackets **<:** and **:>**. Each parameter consists of a keyletter, possibly followed immediately by a value. The following parameters are recognized:

t*tabs* The **t** parameter specifies the tab settings for the file. The value of *tabs* must be one of the following:

1. a list of column numbers separated by commas, indicating tabs set at the specified columns;
2. a – followed immediately by an integer *n*, indicating tabs at intervals of *n* columns;
3. a – followed by the name of a "canned" tab specification.

Standard tabs are specified by **t–8**, or equivalently, **t1,9,17,25,**etc. The canned tabs which are recognized are defined by the *tabs*(1) command.

s*size* The **s** parameter specifies a maximum line size. The value of *size* must be an integer. Size checking is performed after tabs have been expanded, but before the margin is prepended.

m*margin* The **m** parameter specifies a number of spaces to be prepended to each line. The value of *margin* must be an integer.

d The **d** parameter takes no value. Its presence indicates that the line containing the format specification is to be deleted from the converted file.

e The **e** parameter takes no value. Its presence indicates that the current format is to prevail only until another format specification is encountered in the file.

Default values, which are assumed for parameters not supplied, are **t–8** and **m0**. If the **s** parameter is not specified, no size checking is performed. If the first line of a file does not contain a format specification, the above defaults are assumed for the entire file. The following is an example of a line containing a format specification:

```
* <:t5,10,15 s72:> *
```

If a format specification can be disguised as a comment, it is not necessary to code the **d** parameter.

SEE ALSO

ed(1), newform(1), tabs(1) in the *User's Reference Manual.*

NAME

fstab – static information about filesystems

DESCRIPTION

The file */etc/fstab* describes the filesystems used by the local machine. The system administrator can modify it with a text editor. It is read by commands that mount, unmount and check the consistency of filesystems. The file consists of a number of lines of the form:

filesystem directory type options frequency pass

For example:

```
/dev/root   /   efs   rw  0 0
```

Fields are separated by white space; a '#' as the first non-white character indicates a comment.

The entries from this file are accessed using the routines in *getmntent*(3), which returns a structure of the following form:

```
struct mntent {
        char    *mnt_fsname;  /* filesystem name */
        char    *mnt_dir;     /* filesystem path prefix */
        char    *mnt_type;    /* efs, nfs, dbg, or ignore */
        char    *mnt_opts;    /* rw, ro, hard, soft */
        int     mnt_freq;     /* dump frequency, in days */
        int     mnt_passno;   /* parallel fsck pass number */
};
```

This structure is defined in the <mntent.h> include file. To compile and link a program that calls *getmntent*(3), follow the procedures for section (3Y) routines as described in *intro*(3).

The *mnt_dir* field is the full path name of the directory to be mounted on. The *mnt_type* field determines how the *mnt_fsname* and *mnt_opts* fields will be interpreted. Here is a list of the filesystem types currently supported, and the way each of them interprets these fields:

efs *mnt_fsname* must be a block special device (e.g., /dev/root).

dbg *mnt_fsname* should be the /debug directory. See *dbg*(4).

nfs *mnt_fsname* is the path on the server of the directory to be served. (NFS option only).

If the *mnt_type* is specified as **ignore**, then the entry is ignored. This is useful to show disk partitions not currently used. *Mnt_freq* and *mnt_passno* are not supported.

The *mnt_opts* field contains a list of comma-separated option words. Some *mnt_opts* are valid for all filesystem types, while others apply to a specific type only.

Options valid on **efs** and **nfs** filesystems (the default is **rw**):

rw read/write.

ro read-only.

noauto ignore this entry during a **mount –a** command, to allow the definition of *fstab* entries for commonly-used filesystems that should not be automatically mounted.

grpid causes a file created within the filesystem to have the group-ID of its parent directory, not the creating process's group-ID.

Options specific to **efs** filesystems (the default is **fsck**):

raw=*path* the filesystem's raw device pathname (e.g. /dev/rroot).

fsck *fsck*(1M) invoked with no filesystem arguments should check this filesystem.

nofsck *fsck*(1M) should not check this filesystem by default.

lbsize=*n* the number of bytes transferred in each read or synchronous write operation.

The value assigned to the **lbsize** option must be a power of two at least as large as **NPBC** (as defined in */usr/include/sys/param.h*), and no larger than 64K. The current default for **lbsize** is the largest power of two less than or equal to the size of one disk track. An invalid size will cause the mount to fail with the error EINVAL. Note that less than **lbsize** bytes will be transferred if there are not **lbsize** contiguous bytes of the addressed portion of the file on disk.

If the NFS option is installed, the following options are valid for **nfs** filesystems:

bg if the first attempt fails, retry in the background.

fg retry in foreground. (Default)

retry=*n* set number of mount failure retries to *n*. (Default = 10000)

rsize=*n* set read buffer size to *n* bytes. (Default = 8K)

wsize=*n*	set write buffer size to *n* bytes. (Default = 8K)
timeo=*n*	set NFS timeout to *n* tenths of a second. (Default = 7)
retrans=*n*	set number of NFS retransmissions to *n*. (Default = 4)
port=*n*	set server UDP port number to *n*. (Default = 2049)
hard	retry request until server responds. (Default)
soft	return error if server doesn't respond.
intr	allow hard mounts to be interrupted by uncaught fatal signals. (Default)
nointr	don't allow hard mounts to be interrupted.
acregmin=*t*	set the regular file minimum attribute cache timeout to *t* seconds. (Default = 3)
acregmax=*t*	set the regular file maximum attribute cache timeout to *t* seconds. (Default = 60)
acdirmin=*t*	set the directory minimum attribute cache timeout to *t* seconds. (Default = 30)
acdirmax=*t*	set the directory maximum attribute cache timeout to *t* seconds. (Default = 60)
actimeo=*t*	set regular and directory minimum and maximum attribute cache timeouts to *t* seconds.
noac	no attribute caching.
private	do not flush delayed writes on last close of an open file, and use local file and record locking instead of a remote lock manager.

The **bg** option causes *mount* to run in the background if the server's *mountd*(1M) does not respond. *Mount* attempts each request **retry=*n*** times before giving up.

Once the filesystem is mounted, each **nfs** request made waits **timeo=*n*** tenths of a second for a response. If no response arrives, the time-out is multiplied by 2 and the request is retransmitted. When **retrans=*n*** retransmissions have been sent with no reply a **soft** mounted filesystem returns an error on the request and a **hard** mounted filesystem retries the request. Filesystems that are mounted **rw** (read-write) should use the **hard** option. The number of bytes in a read or write request can be set with the **rsize** and **wsize** options.

In the absence of client activity that would invalidate recently acquired file attributes, NFS holds attributes cached for an interval between **acregmin** and **acregmax** for regular files, and between **acdirmin** and **acdirmax** for directories. The **actimeo** option sets all attribute timeout constraints to a given number of seconds. The **noac** option disables attribute caching altogether.

The **private** option greatly improves write performance by caching data and delaying writes on the assumption that only this client modifies files in the remote filesystem. It should be used only if the greater risk of lost delayed-write data in the event of a crash is acceptable given better performance. Note that EFS uses caching strategies similar to private NFS, and that the system reduces the risk of data loss for all filesystems by automatically executing a partial *sync*(2) at regular intervals.

NOTES

The default *fstab* supplied with SGI systems contains the following entry for the /usr filesystem:

```
/dev/usr /usr efs rw,raw=/dev/rusr 0 0
```

The setup program *MAKEDEV* (see *makedev*(1M)) creates */dev/usr* and */dev/rusr* as links to partition 6 on the root disk. This is the normal disk usage; however, if you wish to set up a machine with the /usr filesystem residing elsewhere (for example, on a second disk or on a logical volume, described in *lv*(7M)), the *mnt_fsname* field must be changed to the full pathname of the device where the */usr* filesystem actually resides. If present, the path specified by the **raw** option should also be changed to the corresponding full pathname. For example:

```
/dev/dsk/ips0d1s7 /usr efs rw,raw=/dev/rdsk/ips0d1s7 0 0
```

Note that if this is done, the */dev/usr* and */dev/rusr* devices created by *MAKEDEV* will not point to the device containing the */usr* filesystem, and they should not be referenced.

Caution: do not attempt to reconfigure a system with /usr in a non-default volume by manually recreating these */dev/usr* and */dev/rusr* links and leaving the fstab entry unchanged. While this would work in normal operation, it will lead to incorrect behaviour when installing new software.

FILES

/etc/fstab

SEE ALSO

fsck(1M), mount(1M), mtab(4)
getmntent(3) if the NFS option is installed.
Extensions by Silicon Graphics, Inc.

NAME

gettydefs – speed and terminal settings used by getty

DESCRIPTION

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

NOTE: Customers who need to support terminals that pass 8 bits to the system (as is typical outside the U.S.A.) must modify the entries in **/etc/gettydefs** as described in the **WARNINGS** section.

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

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

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

label This is the string against which *getty*(1M) tries to match its second argument. It is often the speed, such as **1200**, at which the terminal is supposed to run, but it need not be (see below).

initial-flags These flags are the initial *ioctl*(2) settings to which the terminal is to be set if a terminal type is not specified to *getty*(1M). The flags that *getty*(1M) understands are the same as the ones listed in **/usr/include/sys/termio.h** [see *termio*(7)]. Normally only the speed flag is required in the *initial-flags*. *getty*(1M) automatically sets the terminal to raw input mode and takes care of most of the other flags. The *initial-flag* settings remain in effect until *getty*(1M) executes *login*(1).

final-flags These flags take the same values as the *initial-flags* and are set just before *getty*(1M) executes *login*(1). The speed flag is again required. The composite flag **SANE** takes care of most of the other flags that need to be set so that the processor and terminal are communicating in a rational fashion. The other two commonly specified *final-flags* are **TAB3**, so that tabs are sent to the terminal as spaces, and **HUPCL**, so that the line is hung up on the final close.

login-prompt This entire field is printed as the *login-prompt*. Unlike the above fields where white space is ignored (a space, tab or new-line), they are included in the *login-prompt* field.

next-label If this entry does not specify the desired speed, indicated by the user typing a <*break*> character, then *getty*(1M) will search for the entry with *next-label* as its *label* field and set up the terminal for those settings. Usually, a series of speeds are linked together in this fashion, into a closed set; for instance, **2400** linked to **1200**, which in turn is linked to **300**, which finally is linked to **2400**.

If *getty*(1M) is called without a second argument, then the first entry of **/etc/gettydefs** is used, thus making the first entry of **/etc/gettydefs** the default entry. It is also used if *getty*(1M) can not find the specified *label*. If **/etc/gettydefs** itself is missing, there is one entry built into *getty*(1M) which will bring up a terminal at **300** baud.

It is strongly recommended that after making or modifying **/etc/gettydefs**, it be run through *getty*(1M) with the check option to be sure there are no errors.

FILES

/etc/gettydefs

SEE ALSO

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

WARNINGS

To support terminals that pass 8 bits to the system (also, see the **BUGS** section), modify the entries in the **/etc/gettydefs** file for those terminals as follows: add **CS8** to *initial-flags* and replace all occurrences of **SANE** with the values: **BRKINT IGNPAR ICRNL IXON OPOST ONCLR CS8 ISIG ICANON ECHO ECHOK**

An example of changing an entry in **/etc/gettydefs** is illustrated below. All the information for an entry must be on one line in the file.

Original entry:

CONSOLE # B9600 HUPCL OPOST ONLCR # B9600 SANE IXANY TAB3 HUPCL # Console Login: # console

Modified entry:

CONSOLE # B9600 CS8 HUPCL OPOST ONLCR # B9600 BRKINT IGNPAR ICNRL IXON OPOST ONLCR CS8 ISIG ICANON ECHO ECHOK IXANY TAB3 HUPCL # Console Login: # console

This change will permit terminals to pass 8 bits to the system so long as the system is in MULTI-USER state. When the system changes to SINGLE-USER state, the *getty*(1M) is killed and the terminal attributes are lost. So to permit a terminal to pass 8 bits to the system in SINGLE-USER state, after you are in SINGLE-USER state, type (see *stty*(1)):

```
stty –istrip cs8
```

BUGS

8-bit with parity mode is not supported.

NAME

group – group membership file

DESCRIPTION

The **/etc/group** file contains for each group the following information:

- group name
- encrypted password
- numerical group ID
- a comma separated list of all users allowed in the group

For example, the entry for the *sys* group is:

sys::0:root,bin,sys,adm

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 demanded. A "*" in the password field locks the entry.

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.

YELLOW PAGES

If the NFS option is installed, 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 non-null password or group member field, the contents of that field will overide what is contained in the Yellow Pages. The numerical group ID field cannot be overridden.

A group file can also have a line beginning with a minus (–), these entries are used to disallow group entries. There is only one style of – entries: an entry that consists of –*name* means to disallow any subsequent entry (if any) for *name*. These entries will be disallowed regardless of whether the subsequent entry comes from the Yellow Pages or the local group file.

For example, if the following entries

–oldproj
+myproject::bill, steve
+:

appear at the end of a group file, then the group *oldproj* will be ignored if it appears after the entry –*oldproj*. Also, 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*.

FILES

/etc/group

SEE ALSO

crypt(3), newgrp(1M), passwd(1), passwd(4)

BUGS

The *passwd*(1) command won't change group passwords.
Sun Microsystems (YP version)

NAME

hosts – host name data base

DESCRIPTION

The **/etc/hosts** file contains information regarding the known hosts on the network. For each host a single line should be present with the following information:

Internet address
official host name
aliases

Items are separated by any number of blanks and/or tab characters. A "#" indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file.

This file must include entries for the machine's network interfaces, the "localhost" address and a few important machines on the local network. *ifconfig*(1M) uses this file when assigning addresses to the network interfaces.

By default, this file is used by *gethostbyname*(3N) and *gethostbyaddr*(3N) only when the Yellow Pages or the Berkeley name server (*named*(1M)) are not enabled. The system can be configured to use YP, *named*, and/or this file, as described in *resolver*(4) and the *Network Communications Guide*.

If the host is not connected to any network, the file should contain an entry defining the hostname as an alias for the "localhost" entry. For example, if the hostname is IRIS, the /etc/hosts file should contain this line:

```
127.1   localhost.   IRIS
```

Sites connected to the Internet should configure the system to use the name server. This file may be created from the official host data base maintained at the Network Information Center (NIC), though local changes may be required to bring it up to date regarding unofficial aliases and/or unknown hosts. The host data base maintained at NIC is incomplete.

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

FILES

/etc/hosts

SEE ALSO

gethostbyname(3N), ifconfig(1M), named(1M), resolver(4), hostname(5)
The network administration chapters in the *Network Communications Guide*.

NAME

hosts.equiv – list of trusted hosts

DESCRIPTION

The */etc/hosts.equiv* file contains a list of trusted hosts. When an *rlogin*(1C), *rcp*(1C), *rdist*(1C), or *rsh*(1C) request from such a host is made, and the initiator of the request is in */etc/passwd*, then, no further validity checking is done. That is, *rlogin* does not prompt for a password, and *rcp*, *rdist*, and *rsh* complete successfully. So a remote user is "equivalenced" to a local user with the same user name when the remote user is in *hosts.equiv*.

The format of *hosts.equiv* is a list of host names, as in this example:

```
bonnie.sgi.com
clyde.sgi.com
```

A line consisting of a simple host name means that anyone logging in from that host is trusted. Programs scan *hosts.equiv* linearly, and stop at the first hit.

The *.rhosts* file has the same format as *hosts.equiv*. When user *XXX* executes *rlogin* or *rsh*, the *.rhosts* file from *XXX*'s home directory is conceptually concatenated onto the end of *hosts.equiv* for permission checking. In the special case when the user is root, then only the */.rhosts* file is checked.

If an entry in a local user's *.rhosts* file contains a remote host name and a user name separated by white space, the remote user is allowed to log in as the local user. Thus the entry

```
bonnie.sgi.com    faye
```

in warren's *.rhosts* allows faye to log in from bonnie as warren. If this entry was placed in clyde's */etc/hosts.equiv*, then faye could login as anyone who is in clyde's */etc/passwd* file.

FILES

/etc/hosts.equiv
~/.rhosts

SEE ALSO

rcp(1C), rdist(1C), rlogin(1C), rsh(1C), rcmd(3N), rhosts(4)

NAME

inittab – script for the init process

DESCRIPTION

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

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

id:rstate:action:process

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

id This field, of up to four characters, is used to uniquely identify an entry.

rstate This defines the *run-level* in which this entry is to be processed. *Run-levels* effectively correspond to a configuration of processes in the system. That is, each process spawned by *init* is assigned a *run-level* or *run-levels* in which it is allowed to exist. The *run-levels* are represented by the letter **s** (or **S**), or a number ranging from **0** through **6**. As an example, if the system is in *run-level* **1**, only those entries having a **1** in the *rstate* field will be processed. When *init* is requested to change *run-levels*, all processes which do not have an entry in the *rstate* field for the target *run-level* will be sent the warning signal (**SIGTERM**) and allowed a grace period (see *init*(1M) for the length of this grace period), before being forcibly terminated by a kill signal (**SIGKILL**). The *rstate* field can define multiple *run-levels* for a process by selecting more than one *run-level* in any combination from **0–6**, **s** and **S**. If no *run-level* is specified, then the process is assumed to be valid at all *run-levels*. There are three other values, **a**, **b** and **c**, which can appear in the *rstate* field, even though they are not true *run-levels*. Entries which have these characters in the *rstate* field are processed only when the *telinit* [see *init*(1M)] process requests them to be run (regardless of the current *run-level* of the system). They differ

from *run-levels* in that *init* can never enter *run-level* **a**, **b** or **c**. Also, a request for the execution of any of these processes does not change the current *run-level*. Furthermore, a process started by an **a**, **b** or **c** command is not killed when *init* changes levels. They are only killed if their line in **/etc/inittab** is marked **off** in the *action* field, their line is deleted entirely from **/etc/inittab**, or *init* goes into the *SINGLE USER* state.

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

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

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

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

boot The entry is to be processed only at *init*'s boot-time read of the *inittab* file. *Init* is to start the process, not wait for its termination; and when it dies, not restart the process. In order for this instruction to be meaningful, the *rstate* should be the default or it must match *init*'s *run-level* at boot time. This action is useful for an initialization function following a hardware reboot of the system.

bootwait The entry is to be processed the first time *init* goes from single-user to multi-user state after the system is booted. (If **initdefault** is set to **2**, the process will run right after the boot.) *Init* starts the process, waits for its termination and, when it dies, does not restart the process.

powerfail Execute the process associated with this entry only when *init* receives a power fail signal [**SIGPWR**, see *signal*(2)].

powerwait Execute the process associated with this entry only when *init* receives a power fail signal (**SIGPWR**) and wait until it terminates before continuing any processing of *inittab*.

off If the process associated with this entry is currently running, send the warning signal (**SIGTERM**) and wait 20 seconds before forcibly terminating the process via the kill signal (**SIGKILL**). If the process is nonexistent, ignore the entry.

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

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

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

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

NOTES

Strictly speaking, except for comments in the *process* field, there is no comment convention for inittab files. Note in particular that a leading # in itself does not cause a line to be treated as a comment. However, lines not in the

id:rstate:action:process format will be ignored by *init*.

FILES

/etc/inittab

SEE ALSO

exec(2), open(2), signal(2).
getty(1M), init(1M) in the *System Administrator's Reference Manual*.
sh(1), who(1) in the *User's Reference Manual*.

NAME

inode – format of an Extent File System inode

SYNOPSIS

#include <sys/param.h>
#include <sys/inode.h>

DESCRIPTION

An *inode* is the volume data structure used by a file system to implement the abstraction of a file. (This is not to be confused with the *in-core inode* used by the operating system to manage files in use.)

An *inode* contains the type (e.g., plain file, directory, symbolic link, or device file) of the file; its owner, group and public access permissions; the owner and group id numbers; its size in bytes; the number of links (directory references) to the file; and the times of last access and last modification to the file. In addition, there is a file system type-dependent representation of the list of data blocks claimed by the file.

An *inode* under the Extent File System has the following structure.

```
#define EFS_DIRECTEXTENTS 12

/*
 * Extent based file system inode as it appears on disk.
 * The efs inode is 128 bytes long.
 */
struct  efs_dinode {
        ushort  di_mode;        /* type and access permissions */
        short   di_nlink;       /* number of links */
        ushort  di_uid;         /* owner's user id number */
        ushort  di_gid;         /* group's group id number */
        off_t   di_size;        /* number of bytes in file */
        time_t  di_atime;       /* time of last access (to contents) */
        time_t  di_mtime;       /* of last modification (of contents) */
        time_t  di_ctime;       /* of last modification to inode */
        long    di_gen;         /* generation number */
        short   di_numextents;  /* # of extents */
        short   di_unused;      /* UNUSED */
        union {
                extent  di_extents[EFS_DIRECTEXTENTS];
                dev_t   di_dev; /* device for IFCHR/IFBLK */
        } di_u;
};
```

The types *ushort*, *off_t*, *time_t*, and *dev_t* are defined in *types*(5). The *extent* type is defined as follows:

```
typedef struct      extent {
        unsigned int
                ex_magic:8,     /* magic #, must be 0 */
                ex_bn:24,       /* bb # on volume */
                ex_length:8,    /* length of this extent in bb's */
                ex_offset:24;   /* logical file offset in bb's */
} extent;
```

di_mode contains the type of the file (plain file, directory, etc), and its read, write, and execute permissions for the file's owner, group, and public. *di_nlink* contains the number of links to the inode. Correctly formed directories have a minimum of two links: a link in the directory's parent and the '.' link in the directory itself. Additional links may be caused by '..' links from subdirectories. *di_uid* and *di_gid* contain the user id and group id of the file (used to determine which set of access permissions apply: owner, group, or public). *di_size* contains the length of the file in bytes.

di_atime is the time of last access to the file's contents. *di_mtime* is the time of last modification of the file's contents. *di_ctime* is the time of last modification of the inode, as opposed to the contents of the file it represents. These times are given in seconds since the beginning of 1970 GMT.

di_gen is the inode generation number used to sequence instantiations of the inode.

An extent descriptor maps a logical segment of a file to to a physical segment (i.e., extent) on the volume. The physical segment is characterized by a starting address and a length, both in basic blocks (of 512 bytes) and a logical file offset, also in basic blocks.

di_numextents is the number of extents claimed by the file. If less than or equal to *EFS_DIRECTEXTENTS* then the extent descriptors appear directly in the inode as *di_u.di_extents[0 .. di_numextents-1]*. When the number of extents exceeds this range, then *di_u.di_extents[0 .. di_u.di_extents[0].ex_offset-1]* are indirect extents that map blocks holding extent information. There are at most *EFS_DIRECTEXTENTS* indirect extents.

If the inode is a block or character special inode, *di_u.di_numexents* is 0, and *di_u.di_dev* contains a number identifying the device.

FILES

/usr/include/sys/param.h
/usr/include/sys/types.h
/usr/include/sys/inode.h

/usr/include/sys/stat.h

SEE ALSO

stat(2), fs(4), efs(4), types(5).

NAME

ldfcn – common object file access routines

SYNOPSIS

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

DESCRIPTION

The common object file access routines are a collection of functions for reading an object file that is in common object file form. Although the calling program must know the detailed structure of the parts of the object file that it processes, the routines effectively insulate the calling program from knowledge of the overall structure of the object file.

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 non-zero.

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's accessed through the pCHDR structure (see *<cmplrs/stsupport.h>*). If no symbol table is present, this field is NULL. **NOTE:** 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:** If you use *libmld.a* routines, all structures, except the optional header and auxiliaries, are swapped.

The object file access functions can be divided into five 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 an object file at (seek to) 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 section of the the symbol table to be read

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

Ldopen and *ldaopen* 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)
FREAD((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 this manual.)

The program must be loaded with the object file access routine library *libmld.a*.

WARNINGS

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.

When applied to object files in an archive **FSEEK** (ldptr,offset,BEGINNING) uses (and **FTELL** returns) an offset relative to the beginning of an individual object file, not the absolute file locations used by *fseek* (stream,offset,0) (and returned by *ftell*).

SEE ALSO

Assembly Language Programmer's Guide
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 – file header for implementation-specific constants

SYNOPSIS

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

DESCRIPTION

The header file *<limits.h>* is a list of *minimum* magnitude limitations imposed by the IRIX operating system. In some cases, the actual values may be greater, and can be obtained at runtime via sysconf() or pathconf() system calls (depending upon the variable desired). In this way a program can use, for example, dynamic memory allocation to utilize greater-than-default limits. See sysconf(2) and pathconf(2). *<limits.h>* also 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 (those prefixed by **LDBL_**) that appear in *<float.h>* are not specified because long doubles are not implemented.

All values in *<limits.h>* and *<float.h>* are specified in decimal.

The file *<limits.h>* contains:

```
#define ARG_MAX      10240                /* max length of arguments to exec */
#define CHAR_BIT     8                    /* # of bits in a "char" */
#define CHAR_MAX     255                  /* max integer value of a "char" */
#define CHAR_MIN     0                    /* min integer value of a "char" */
#define CHILD_MAX    25                   /* max # of processes per user id */
#define CLK_TCK      100                  /* # of clock ticks per second */
#define DBL_DIG      15                   /* digits of precision of a "double" */
#define DBL_MAX      1.797693134862316e+308  /* max decimal value of a "double" */
#define DBL_MIN      2.225073858507201e-308  /* min decimal value of a "double" */
#define FCHR_MAX     2147483647           /* max size of a file in bytes */
#define FLT_DIG      6                    /* digits of precision of a "float" */
#define FLT_MAX      3.40282347e+38       /* max decimal value of a "float" */
#define FLT_MIN      1.17549435e-38       /* min decimal value of a "float" */
#define HUGE_VAL     1.797693134862316e+308  /* error value returned by Math lib */
#define INT_MAX      2147483647           /* max decimal value of an "int" */
#define INT_MIN      -2147483648          /* min decimal value of an "int" */
#define LINK_MAX     1000                 /* max # of links to a single file */
#define LONG_MAX     2147483647           /* max decimal value of a "long" */
#define LONG_MIN     -2147483648          /* min decimal value of a "long" */
#define MAX_CANON    255                  /* max # of bytes in a terminal */
                                          /* canonical input queue */
#define MAX_INPUT    255                  /* max # of bytes for which space will be */
```

```
                                        /* available in a terminal input queue */
#define MB_LEN_MAX      1               /* max # of characters in a multibyte */
                                        /* character */
#define NAME_MAX        255             /* max # of characters in a file name */
#define NGROUPS_MAX     0               /* max # of simultaneous supplementary
                                        /* group IDs per process */
#define OPEN_MAX        20              /* max # of files a process can have open
#define PASS_MAX        8               /* max # of characters in a password */
#define PATH_MAX        1024            /* max # of characters in a path name */
#define PID_MAX         30000           /* max value for a process ID */
#define PIPE_BUF        10240           /* max # bytes atomic in write to a pipe *
#define PIPE_MAX        10240           /* max # bytes written to a pipe in a writ
#define SCHAR_MAX       127             /* max decimal value of a "signed char" *
#define SCHAR_MIN       -128            /* min decimal value of a "signed char" *
#define SHRT_MAX        32767           /* max decimal value of a "short" */
#define SHRT_MIN        -32768          /* min decimal value of a "short" */
#define STD_BLK         1024            /* # bytes in a physical I/O block */
#define SYS_NMLN        9               /* # of chars in uname-returned strings */
#define UCHAR_MAX       255             /* max decimal value of an "unsigned ch
#define UID_MAX         60000           /* max value for a user or group ID */
#define UINT_MAX        4294967295      /* max decimal value of an "unsigned int
#define ULONG_MAX       4294967295      /* max decimal value of an "unsigned lon
#define USHRT_MAX       65535           /* max decimal value of an "unsigned sh
#define USI_MAX         4294967295      /* max decimal value of an "unsigned" */
#define WORD_BIT        32              /* # of bits in a "word" or "int" */
```

POSIX additions to *<limits.h>*. The POSIX 1003.1 standard requires the following symbols to be defined in *<limits.h>*, with the values shown. These define *minimum* values for certain features of the system; hence no POSIX 1003.1 conforming system can provide a more restrictive value. For each of these symbols, there is an analogous symbol defined in *<limits.h>*, which reflects the actual implementation, and which are, in most cases, less restrictive.

```
#define _POSIX_ARG_MAX      4096    /* Minimum value for ARG_MAX */
#define _POSIX_CHILD_MAX    6       /* Minimum value for CHILD_MAX */
#define _POSIX_LINK_MAX     8       /* Minimum value for LINK_MAX */
#define _POSIX_MAX_CANON    255     /* Minimum value for MAX_CANON *
#define _POSIX_MAX_INPUT    255     /* Minimum value for MAX_INPUT */
#define _POSIX_NAME_MAX     14      /* Minimum value for NAME_MAX */
#define _POSIX_NGROUPS_MAX  0       /* Minimum value for NGROUPS_MA
#define _POSIX_OPEN_MAX     16      /* Minimum value for OPEN_MAX */
#define _POSIX_PATH_MAX     255     /* Minimum value for PATH_MAX */
#define _POSIX_PIPE_BUF     512     /* Minimum value for PIPE_BUF */
```

The file *<float.h>* contains:

```
#define FLT_RADIX       2           /* radix of exponent representation */
#define FLT_ROUNDS      1           /* addition rounds (>0 implemented) */

/* 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 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         2.225073858507201e-308

/* minimum negative integer such that 10 raised to that power */
/* is in 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         1.797693134862316e+308

/* maximum integer such that 10 raised to that power is in the */
/* range of finite floating-point numbers */
#define FLT_MAX_10_EXP  38
```

```
#define DBL_MAX_10_EXP      308
```

SEE ALSO

sysconf(2), pathconf(2)

NAME

linenum – line number entries in a MIPS object file

SYNOPSIS

#include <sym.h>

DESCRIPTION

The *cc*(1), *f77*(1), *pc*(1), and *pl1*(1) commands generate an entry in the object file for each source line on which a breakpoint is possible [when any of the commands are invoked with the **–g** option]. Users can then reference line numbers when using the appropriate software test system (see *edge*(1) and *dbx*(1)). The structure of these line number entries is described in the *Assembly Language Programmer's Guide.*

NOTE

Do not include *<linenum.h>;* the structures for dealing with line numbers in *<sym.h>* supercede those in *<linenum.h>*.

SEE ALSO

Assembly Language Programmer's Guide.
cc(1), edge(1), dbx(1), f77(1), pc(1), pl1(1), a.out(4), syms(4).

NAME

login – login configuration file

SYNOPSIS

/etc/config/login.options

DESCRIPTION

login.options is an ASCII file consisting of lines of the form **keyword** or **keyword**=*value*. Keywords can be separated by white-space or placed on separate lines. Keywords that take values must be one word with no white-space between the keyword, equals sign and value. A "#" indicates the beginning of a comment; characters up to the end of the line are ignored.

The following keywords are recognized:

maxtries=*value*

The number of unsuccessful attempts permitted before ending the session. 0 is "no limit". The default is 5 tries.

disabletime=*value*

The amount of time in seconds *login* waits before ending the session after **maxtries** unsuccessful attempts. The default is 20 seconds.

passwdreq

All accounts must have passwords. If the user does not have a password, the user will be forced to choose one before being allowed to login.

lastlog

Inform the user about the last successful login attempt. It shows the date, time and the name of the terminal or remote host from which the previous login attempt occurred.

syslog=*value*

Record successful and unsuccessful login attempts to *syslog*(3) if *value* is **all** or record unsuccessful attempts only if *value* is **fail.**

FILES

/etc/config/login.options

SEE ALSO

getty(1M),
login(1) in the *User's Reference Manual.*

NAME

lvtab – information about logical volumes

DESCRIPTION

The file */etc/lvtab* describes the logical volumes used by the local machine. There is an entry in this file for every logical volume which will be used by the machine. It is read by commands that create, install and check the consistency of logical volumes. The system administrator can modify it with a text editor to add new logical volumes or to extend existing ones.

The file consists of entries which have the form:

volume_device_name:[volume_name]:[options]:device_pathnames

For example:

```
lv0:logical volume test:stripes=3:devs=/dev/dsk/ips0d1s7,\
/dev/dsk/ips0d2s7, /dev/dsk/ips0d3s7
```

Fields are separated by colons, and lines may be continued by the usual backslash convention as illustrated above. A '#' as the first non-white character indicates a comment; blank lines may be present in the file and will be ignored.

The fields in each entry have the following significance:

volume_device_name

This indicates the names of the special files through which the system will access the logical volume. In the above example, the entry *lv0* implies that the logical volume will be accessed via the device special files /dev/dsk/lv0 and /dev/rdsk/lv0. Note that volume device names are expected to be of the form 'lv' followed by one or 2 digits; this is enforced by the logical volume utilities.

volume name

This is a human-readable identifying name for the logical volume. The logical volume labels on the disks constituting a volume also carry a copy of the volume name, so utilities are able to check that the logical volume on the disks physically present is actually the volume expected by /etc/lvtab.

This field may be null (indicated by a second colon immediately following the one terminating the *volume_device_name* field). This is legal but deprecated, since in this case, no identity check of the logical volume can be done by the utilities.

options Some numerical options concerning the volume may appear. These are specified in the format "option_name=number". There must be no space between the option_name, the '=' sign, and the numerical value given. Options are separated by colons, as with other fields in an entry.

Currently recognized options are:

stripes=
step=

The stripes option allows a striped logical volume to be created; the value of the parameter specifies the number of ways the volume storage is striped across its constituent devices. If this option is omitted, the logical volume is unstriped.

The step option is meaningful only for striped volumes (and is ignored otherwise); it specifies the granularity with which the storage is to be round-robin distributed over the constituent devices. If this option is omitted, the default is a step of the device tracksize; this is generally a good value so the step option is not normally needed.

device_pathnames

Following any numerical options, there must be a list of the block special file pathnames of the devices constituting the logical volume. This is introduced by the keyword

devs=

The pathnames must be comma-separated.
Each pathname should be the name of the special file for a disk device partition in the /dev/dsk directory. The partition must be one which is legal for use as normal data storage, ie. it must not be one of the dedicated partitions such as the disk volume label, track replacement area etc.

Note that if the volume is striped, some restrictions apply: the number of pathnames must be a multiple of **stripes**. Further, considering the pathnames as successive groups, each of **stripes** pathnames, the devices in each group must be all of the same size, and must have the same number of sectors per track.
To obtain best performance from striping, each disk (within every group of 'stripes' disks) should be on a separate controller.

The entries from this file are accessed using the routines in *getlvent*(3), which returns a structure of the following form:

```
struct lvtabent     {
        char            *devname;       /* volume device name */
        char            *volname;       /* volume name (human-readable) */
        unsigned stripe;        /* number of ways striped */
        unsigned gran;          /* granularity of striping */
        unsigned ndevs;         /* number of constituent devices */
        int             mindex;         /* not currently used. */
        char            *pathnames[1];  /* pathnames of constituent devices */
};
```

This structure is defined in the <lvtab.h> include file.

FILES

/etc/lvtab

SEE ALSO

lvinit(1M), mklv(1M), lvck(1M), getlvent(3), lv(7M).

NAME

master – master configuration database

DESCRIPTION

The *master* configuration database is a collection of files. Each file contains configuration information for a device or module that may be included in the system. A file is named with the module name to which it applies. This collection of files is maintained in a directory called **/usr/sysgen/master.d**. Each individual file has an identical format. For convenience, this collection of files will be referred to as the *master* file, as though it was a single file. This will allow a reference to the *master* file to be understood to mean the *individual file* in the **master.d** directory that corresponds to the name of a device or module. The file is used by the *lboot*(1M) program to obtain device information to generate the device driver and configurable module files. *master* consists of two parts; they are separated by a line with a dollar sign ($) in column 1. Part 1 contains device information for both hardware and software devices, and loadable modules. Part 2 contains parameter declarations. Any line with an asterisk (*) in column 1 is treated as a comment.

Part 1, Description

Hardware devices, software drivers and loadable modules are defined with a line containing the following information. Field 1 must begin in the left most position on the line. Fields are separated by white space (tab or blank).

Field 1: element characteristics:

o	specify only once
r	required device
b	block device
c	character device
t	initialize cdevsw[].d_ttys
j	file system
s	software driver
f	STREAMS driver
m	STREAMS module
x	not a driver; a loadable module
k	kernel module
n	driver is fully semaphored for multiprocessor operation; the **n, p** and **l** directives are ignored on single-processor systems

	p	driver is not semaphored and should run on only one processor
	q	driver is not semaphored and should run on network processor
Field 2:	handler prefix (14 chars. maximum)	
Field 3:	software driver external major number; "–" if not a software driver, or to be assigned during execution of *lboot*(1M)	
Field 4:	number of sub-devices per device; "–" if none	
Field 5:	dependency list (optional); this is a comma separated list of other drivers or modules that must be present in the configuration if this module is to be included	

For each module, two classes of information are required by *lboot*(1M): external routine references and variable definitions. Routine lines begin with white space and immediately follow the initial module specification line. These lines are free form, thus they may be continued arbitrarily between non-blank tokens as long as the first character of a line is white space. Variable definition lines begin after a line that contains a '$' in column one. Variable definitions follow C language conventions, with slight modifications.

Part 1, Routine Reference Lines

If the UNIX system kernel or other dependent module contains external references to a module, but the module is not configured, then these external references would be undefined. Therefore, the *routine reference* lines are used to provide the information necessary to generate appropriate dummy functions at boot time when the driver is not loaded.

Routine references are defined as follows:

Field 1:	routine name ()
Field 2:	the routine type: one of
	{} routine_name(){}
	{nulldev} routine_name(){nulldev();}
	{nosys} routine_name(){return nosys();}
	{nodev} routine_name(){return nodev();}
	{false} routine_name(){return 0;}
	{true} routine_name(){return 1;}

```
{fsnull} routine_name(){return fsnull();}
{fsstray}
        routine_name(){return fsstray();}
{nopkg}
        routine_name(){nopkg();}
{noreach}
        routine_name(){noreach();}
```

Part 2, Variables

Variables may be declared and (optionally) statically initialized on lines after a line whose first character is a dollar sign ('$'). Variable definitions follow standard C syntax for global declarations, with the following in–line substitutions:

##M	the internal major number assigned to the current module if it is a device driver; zero if this module is not a device driver
##E	the external major number assigned to the current module; either explicitly defined by the current master file entry, or assigned by lboot(1M)
##C	number of controllers present; this number is determined dynamically by lboot(1M) for hardware devices, or by the number provided in the system file for non-hardware drivers or modules
##D	number of devices per controller taken directly from the current master file entry

EXAMPLES

A sample *master* file for a shared memory module would be named "**shm**". The module is an optional loadable software module that can only be specified once. The module prefix is **shm**, and it has no major number associated with it. In addition, another module named "*ipc*" is necessary for the correct operation of this module.

```
* FLAG PREFIX SOFT #DEV DEPENDENCIES
ox  shm  -  -  ipc
                        shmsys(){nosys}
                        shmexec(){}
                        shmexit(){}
                        shmfork(){}
                        shmslp(){true}
                        shmtext(){}
$
```

```
#define SHMMAX 131072
#define SHMMIN 1
#define SHMMNI 100
#define SHMSEG 6
#define SHMALL 512

struct shmid_ds shmem[SHMMNI];
struct shminfo shminfo = {
    SHMMAX,
    SHMMIN,
    SHMMNI,
    SHMSEG,
    SHMALL,
};
```

This *master* file will cause routines named *shmsys*, *shmexec*, etc., to be generated by the boot program if the **shm** driver is not loaded, and there is a reference to this routine from any other module loaded. When the driver is loaded, the structure array *shmem* will be allocated, and the structure *shminfo* will be allocated and initialized as specified.

A sample *master* file for a VME disk driver would be named "**dkip**" The driver is a block and a character device, the driver prefix is **dkip**, and the external major number is 4. The VME interrupt priority level and vector numbers are declared in the system file */usr/sysgen/system* (see lboot(1M)).

```
* FLAG PREFIX SOFT #DEV DEPENDENCIES
bc   dkip 4  -  -  io

$$$
/* disk driver variable tables */
#include "sys/dvh.h"
#include "sys/dkipreg.h"
#include "sys/elog.h"

struct iotime dkipiotime[##C][DKIPUPC];     /* io statistics */
struct iobuf dkipctab[##C];                 /* controller queues */
struct iobuf dkiputab[##C][DKIPUPC];        /* drive queues */
int dkipmajor = ##E;                        /* external major # */
```

This *master* file will cause entries in the block and character device switch tables to be generated, if this module is loaded. Since this is a hardware device (implied by the block and character flags), VME interrupt structures

will be generated, also, by the boot program. The declared arrays will all be sized to the number of controllers present, which is determined by the boot program, based on information in the system file */usr/sysgen/system.*

FILES

/usr/sysgen/master.d/*
/usr/sysgen/system

SEE ALSO

system(4), lboot(1M)

NAME

motd – message of the day

DESCRIPTION

The file **/etc/motd** contains information intended to be displayed on the terminal at login time. For *sh*(1) users, this function is performed by the script **/etc/profile**. For *csh*(1) users, the **/etc/cshrc** displays the contents of the message of the day file.

FILES

/etc/motd

SEE ALSO

csh(1), login(1), sh(1), cshrc(4), profile(4).

NAME

mtab – mounted file system table

DESCRIPTION

Mtab resides in the */etc* directory, and contains a table of filesystems currently mounted by the *mount* command. *Umount* removes entries from this file.

The file contains a line of information for each mounted filesystem, structurally identical to the contents of */etc/fstab*, described in *fstab*(4). There are a number of lines of the form:

fsname dir type opts freq passno

For example:

/dev/root / efs rw 0 0

The file is accessed by programs using *getmntent*(3), and by the system administrator using a text editor.

FILES

/etc/mtab

SEE ALSO

mount(1M), fstab(4)
getmntent(3) if the NFS option is installed.

NAME

networks – network name data base

DESCRIPTION

The **/etc/networks** file contains information regarding the known networks which comprise the DARPA Internet. For each network a single line should be present with the following information:

official network name
network number
aliases

Items are separated by any number of blanks and/or tab characters. A "#" indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which 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 and/or unknown networks.

Network number may be specified in the conventional "." 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.

YELLOW PAGES

If the NFS option is installed and Yellow Pages is running, the *getnetent*(3N) library routines do not access this file.

FILES

/etc/networks

SEE ALSO

getnetent(3N)

BUGS

A name server should be used instead of a static file. A binary indexed file format should be available for fast access.

NAME

passwd – password file

DESCRIPTION

The **/etc/passwd** file contains the following information for each user :

name — User's login name — contains no upper case characters and must not be greater than eight characters long.

password — Encrypted password and optional password aging information

numerical user ID
: This is the user's ID in the system and it must be unique.

numerical group ID
: This is the number of the group that the user belongs to.

user's real name
: In some versions of UNIX, this field also contains the user's office, extension, home phone, and so on. For historical reasons this field is called the GECOS field.

initial working directory
: The directory that the user is positioned in when they log in — this is known as the 'home' directory.

shell — program to use as Shell when the user logs in.

An entry beginning with # is ignored. The user's real name field may contain '&', meaning insert the login name.

The password file 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, */bin/sh* is used.

Password aging is effected for a particular user if his encrypted password is followed by a comma and a non-null string of characters from a 64-character alphabet (**.,/,0-9, A-Z, a-z**). The first character of the age, *M* say, denotes the maximum number of weeks for which a password is valid. A user who attempts to login after his password has expired will be forced to change his password. The next character, *m* say, denotes the minimum period in weeks which must expire before the password may be changed. If the second character is omitted, zero week is the default minimum. *M* and *m* have numerical values in the range 0-63 that correspond to the 64-character alphabet shown above (i.e., / = 1 week; **z** = 63 weeks). If $m = M = 0$ (derived from the string . or ..) the user will be forced to change his password the next time he logs in (and the "age" will disappear from his entry in the password file). If $m > M$ (signified, e.g., by the string ./) only the superuser will be able to change the password.

The password file resides in the */etc* directory. Because of the encrypted passwords, it has general read permission and can be used, for example, to map numerical user ID's to names.

The **passmgmt** command can be used to update information in the */etc/passwd* file. Appropriate precautions must be taken to lock the */etc/passwd* file against simultaneous changes if it is to be edited with a text editor.

YELLOW PAGES

If the NFS option is installed, the *passwd* file can also have a line beginning with a plus (+), which means to incorporate entries from the Yellow Pages. There are three styles of + entries: all 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 non-null password, directory, GECOS, or shell field, they will override what is contained in the Yellow Pages. The numerical user ID and group ID fields cannot be overridden.

Here is a sample */etc/passwd* file:

```
root:q.mJzTnu8icF.:0:10:superuser:/:/bin/csh
bill:6k/7KCFRPNVXg,z/:508:10:Bill The Cat:/usr2/bill:/bin/csh
+john:
+@documentation:no-login:
+:::Guest
```

In this example, there are specific entries for users *root* and *bill*, in case the Yellow Pages are not functioning. The user *bill* will have 63 weeks of maximum password aging and 1 week of minimum password aging. 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

getpwent(3), passmgmt(1M), login(1), crypt(3), a64l(3C), passwd(1), group(4), netgroup(4)

NAME

pnch – file format for card images

DESCRIPTION

The PNCH format is a convenient representation for files consisting of card images in an arbitrary code.

A PNCH file is a simple concatenation of card records. A card record consists of a single control byte followed by a variable number of data bytes. The control byte specifies the number (which must lie in the range 0-80) of data bytes that follow. The data bytes are 8-bit codes that constitute the card image. If there are fewer than 80 data bytes, it is understood that the remainder of the card image consists of trailing blanks.

NAME

printcap – printer capability data base

SYNOPSIS

/etc/printcap

DESCRIPTION

Printcap is a simplified version of the *termcap*(4) data base used to describe line printers. The spooling system accesses the *printcap* file every time it is used, allowing dynamic addition and deletion of printers. Each entry in the data base is used to describe one printer. This data base may not be substituted for, as is possible for *termcap*, because it may allow accounting to be bypassed.

The default printer is normally *lp*, though the environment variable PRINTER may be used to override this. Each spooling utility supports an option, –P*printer*, to allow explicit naming of a destination printer.

Refer to the *4.3BSD Line Printer Spooler Manual* for a complete discussion on how setup the database for a given printer.

CAPABILITIES

Refer to *termcap*(4) for a description of the file layout.

Name	Type	Default	Description
af	str	NULL	name of accounting file
br	num	none	if lp is a tty, set the baud rate (ioctl call)
cf	str	NULL	cifplot data filter
df	str	NULL	tex data filter (DVI format)
fc	num	0	if lp is a tty, clear flag bits (see compatibility notes)
ff	str	"\f"	string to send for a form feed
fo	bool	false	print a form feed when device is opened
fs	num	0	like 'fc' but set bits (see compatibility notes)
gf	str	NULL	graph data filter (plot (3X) format)
hl	bool	false	print the burst header page last
ic	bool	false	driver supports (non standard) ioctl to indent printout
if	str	NULL	name of text filter which does accounting
lf	str	"/dev/console"	error logging file name
lo	str	"lock"	name of lock file
lp	str	"/dev/lp"	device name to open for output

Name	Type	Default	Description
mx	num	1000	maximum file size in BUFSIZ blocks, zero = unlimited
nd	str	NULL	next directory for list of queues (unimplemented)
nf	str	NULL	ditroff data filter (device independent troff)
of	str	NULL	name of output filtering program
pc	num	200	price per foot or page in hundredths of cents
pl	num	66	page length (in lines)
pw	num	132	page width (in characters)
px	num	0	page width in pixels (horizontal)
py	num	0	page length in pixels (vertical)
rf	str	NULL	filter for printing FORTRAN style text files
rg	str	NULL	restricted group. Only members of group allowed access
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 and 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 burst 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
vf	str	NULL	raster image filter
xc	num	0	if lp is a tty, clear local mode bits (tty (4))
xs	num	0	like 'xc' but set bits

If the local line printer driver supports indentation, the daemon must understand how to invoke it.

FILTERS

The *lpd*(1M) daemon creates a pipeline of *filters* to process files for various printer types. The filters selected depend on the flags passed to *lpr*(1). The pipeline set up is:

–p	pr | if	regular text + *pr*(1)
none	if	regular text
–c	cf	cifplot
–d	df	DVI (tex)

–g	gf	*plot*(3)
–n	nf	ditroff
–f	rf	Fortran
–t	tf	troff
–v	vf	raster image

The **if** filter is invoked with arguments:

if [**–c**] –wwidth –llength –iindent **–n** login **–h** host acct-file

The **–c** flag is passed only if the **–l** flag (pass control characters literally) is specified to *lpr*. *Width* and *length* specify the page width and length (from **pw** and **pl** respectively) in characters. The **–n** and **–h** parameters specify the login name and host name of the owner of the job respectively. *Acct-file* is passed from the **af** *printcap* entry.

If no **if** is specified, **of** is used instead, with the distinction that **of** is opened only once, while **if** is opened for every individual job. Thus, **if** is better suited to performing accounting. The **of** is only given the *width* and *length* flags.

All other filters are called as:

filter –xwidth –ylength **–n** login **–h** host acct-file

where *width* and *length* are represented in pixels, specified by the **px** and **py** entries respectively.

All filters take *stdin* as the file, *stdout* as the printer, may log either to *stderr* or using *syslog*(3), and must not ignore SIGINT.

LOGGING

Error messages generated by the line printer programs themselves (that is, the *lp** programs) are logged by *syslog*(3) using the *LPR* facility. Messages printed on *stderr* of one of the filters are sent to the corresponding **lf** file. The filters may, of course, use *syslog* themselves.

Error messages sent to the console have a carriage return and a line feed appended to them, rather than just a line feed.

COMPATIBILITY NOTES

In an attempt to provide compatibility with existing BSD printcap entries, the SGI version of the *lpd* spooler emulates the output bits in the BSD tty flag word (defined in the BSD include file <sgtty.h>) via IRIX termio.

SEE ALSO

termcap(4), lpc(1M), lpd(1M), pac(1M), lpr(1), lpq(1), lprm(1)

NAME

profile – setting up an environment at login time

SYNOPSIS

/etc/profile
$HOME/.profile

DESCRIPTION

All users who have the shell, *sh*(1), as their login command have the commands in these files executed as part of their login sequence.

/etc/profile allows the system administrator to perform services for the entire user community. Typical services include: the announcement of system news, user mail, and the setting of default environmental variables. It is not unusual for */etc/profile* to execute special actions for the **root** login or the *su*(1) command.

The file *$HOME/.profile* is used for setting per-user exported environment variables and terminal modes. The following example is typical (except for the comments):

```
# Set the file creation mask to prohibit
# others from reading my files.
umask 027
# Add my own /bin directory to the shell search sequence.
PATH=$PATH:$HOME/bin
# Set terminal type
eval 'tset -S -Q'
# Set the interrupt character to control-c.
stty intr ^c
# List directories in columns if standard out is a terminal.
ls()  { if [ -t ]; then /bin/ls -C $*; else /bin/ls $*; fi }
```

FILES

/etc/TIMEZONE	timezone environment
$HOME/.profile	user-specific environment
/etc/profile	system-wide environment

SEE ALSO

terminfo(4), timezone(4), environ(5), term(5).
env(1), login(1), mail(1), sh(1), stty(1), tset(1), tput(1) in the *User's Reference Manual.*
su(1M) in the *System Administrator's Reference Manual.*
User's Guide.
Chapter 9 in the *Programmer's Guide.*

NOTES

Care must be taken in providing system-wide services in */etc/profile*. Personal *.profile* files are better for serving all but the most global needs.

NAME

proto – prototype job file for at

SYNOPSIS

/usr/lib/cron/.proto

/usr/lib/cron/.proto.*queue*

DESCRIPTION

When a job is submitted to *at*(1) or *batch*(1), the job is constructed as a shell script. First, a prologue is constructed, consisting of:

- A header whether the job is an *at* job or a *batch* job (actually, *at* jobs submitted to all queues other than queue **a**, not just to the batch queue **b**, are listed as *batch* jobs); the header will be

  : at job

  for an *at* job, and

  : batch job

  for a *batch* job.

- A set of Bourne shell commands to make the environment (see *environ*(5)) for the *at* job the same as the current environment;

- A command to run the user's shell (as specified by the SHELL environment variable) with the rest of the job file as input.

At then reads a "prototype file," and constructs the rest of the job file from it.

Text from the prototype file is copied to the job file, except for special "variables" that are replaced by other text:

$d	is replaced by the current working directory
$l	is replaced by the current file size limit (see *ulimit*(2))
$m	is replaced by the current umask (see *umask*(2))
$t	is replaced by the time at which the job should be run, expressed as seconds since January 1, 1970, 00:00 Greenwich Mean Time, preceded by a colon
$<	is replaced by text read by *at* from the standard input (that is, the commands provided to *at* to be run in the job)

If the job is submitted in queue *queue*, *at* uses the file **/usr/lib/cron/.proto.*queue*** as the prototype file if it exists, otherwise it will use the file **/usr/lib/cron/.proto.**

EXAMPLES

The standard **.proto** file supplied is:

```
#ident  "@(#)adm:.proto         1.2"
cd $d
ulimit $l
umask $m
$<
```

which causes commands to change the current directory in the job to the current directory at the time *at* was run, to change the file size limit in the job to the file size limit at the time *at* was run, and to change the umask in the job to the umask at the time *at* was run, to be inserted before the commands in the job.

FILES

/usr/lib/cron/.proto
/usr/lib/cron/.proto.*queue*

SEE ALSO

at(1)

NAME

protocols – protocol name data base

DESCRIPTION

The **/etc/protocols** file 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

Items are separated by any number of blanks and/or tab characters. A "#" indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file.

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

YELLOW PAGES

If the NFS option is installed and Yellow Pages is running, the *getprotoent*(3N) library routines do not access this file.

FILES

/etc/protocols

SEE ALSO

getprotoent(3N)

BUGS

A name server should be used instead of a static file.

NAME

queuedefs – at/batch/cron queue description file

SYNOPSIS

/usr/lib/cron/queuedefs

DESCRIPTION

The *queuedefs* file describes the characteristics of the queues managed by *cron*(1M). Each non-comment line in this file describes one queue. The format of the lines are as follows:

q.[*njob***j**][*nice***n**][*nwait***w**]

The fields in this line are:

q The name of the queue. **a** is the default queue for jobs started by *at*(1); **b** is the default queue for jobs started by *batch*(1); **c** is the default queue for jobs run from a **crontab** file.

njob The maximum number of jobs that can be run simultaneously in that queue; if more than *njob* jobs are ready to run, only the first *njob* jobs will be run, and the others will be run as jobs that are currently running terminate. The default value is 100.

nice The *nice*(1) value to give to all jobs in that queue that are not run with a user ID of super-user. The default value is 2.

nwait The number of seconds to wait before rescheduling a job that was deferred because more than *njob* jobs were running in that job's queue, or because more than 25 jobs were running in all the queues. The default value is 60.

Lines beginning with # are comments, and are ignored.

EXAMPLE

a.4j1n
b.2j2n90w

This file specifies that the **a** queue, for *at* jobs, can have up to 4 jobs running simultaneously; those jobs will be run with a *nice* value of 1. As no *nwait* value was given, if a job cannot be run because too many other jobs are running *cron* will wait 60 seconds before trying again to run it. The **b** queue, for *batch* jobs, can have up to 2 jobs running simultaneously; those jobs will be run with a *nice* value of 2. If a job cannot be run because too many other jobs are running, *cron* will wait 90 seconds before trying again to run it. All other queues can have up to 100 jobs running simultaneously; they will be run with a *nice* value of 2, and if a job cannot be run because

too many other jobs are running *cron* will wait 60 seconds before trying again to run it.

FILES

/usr/lib/cron/queuedefs

SEE ALSO

cron(1M)

NAME

rcsfile – format of RCS file

DESCRIPTION

An RCS file is an ASCII file. Its contents are described by the grammar below. The text is free format, i.e., spaces, tabs and new lines have no significance except in strings. Strings are enclosed by '@'. If a string contains a '@', it must be doubled.

The meta syntax uses the following conventions: '|' (bar) separates alternatives; '{' and '}' enclose optional phrases; '{' and '}*' enclose phrases that may be repeated zero or more times; '{' and '}+' enclose phrases that must appear at least once and may be repeated; '<' and '>' enclose nonterminals.

<rcstext>	::=	<admin> {<delta>}* <desc> {<deltatext>}*	
<admin>	::=	**head**	{<num>};
		access	{<id>}*;
		symbols	{<id> : <num>}*;
		locks	{<id> : <num>}*;
		comment	{<string>};
<delta>	::=	<num>	
		date	<num>;
		author	<id>;
		state	{<id>};
		branches	{<num>}*;
		next	{<num>};
<desc>	::=	**desc**	<string>
<deltatext>	::=	<num>	
		log	<string>
		text	<string>
<num>	::=	{<digit>{.}}+	
<digit>	::=	0 | 1 | ... | 9	
<id>	::=	<letter>{<idchar>}*	
<letter>	::=	A | B | ... | Z | a | b | ... | z	
<idchar>	::=	Any printing ASCII character except space, tab, carriage return, new line, and <special>.	

<special> ::= ; | : | , | @

<string> ::= @{any ASCII character, with '@' doubled}*@

Identifiers are case sensitive. Keywords are in lower case only. The sets of keywords and identifiers may overlap.

The <delta> nodes form a tree. All nodes whose numbers consist of a single pair (e.g., 2.3, 2.1, 1.3, etc.) are on the "trunk", and are linked through the "next" field in order of decreasing numbers. The "head" field in the <admin> node points to the head of that sequence (i.e., contains the highest pair).

All <delta> nodes whose numbers consist of 2n fields (n≥2) (e.g., 3.1.1.1, 2.1.2.2, etc.) are linked as follows. All nodes whose first (2n)-1 number fields are identical are linked through the "next" field in order of increasing numbers. For each such sequence, the <delta> node whose number is identical to the first 2(n-1) number fields of the deltas on that sequence is called the branchpoint. The "branches" field of a node contains a list of the numbers of the first nodes of all sequences for which it is a branchpoint. This list is ordered in increasing numbers.

Example:

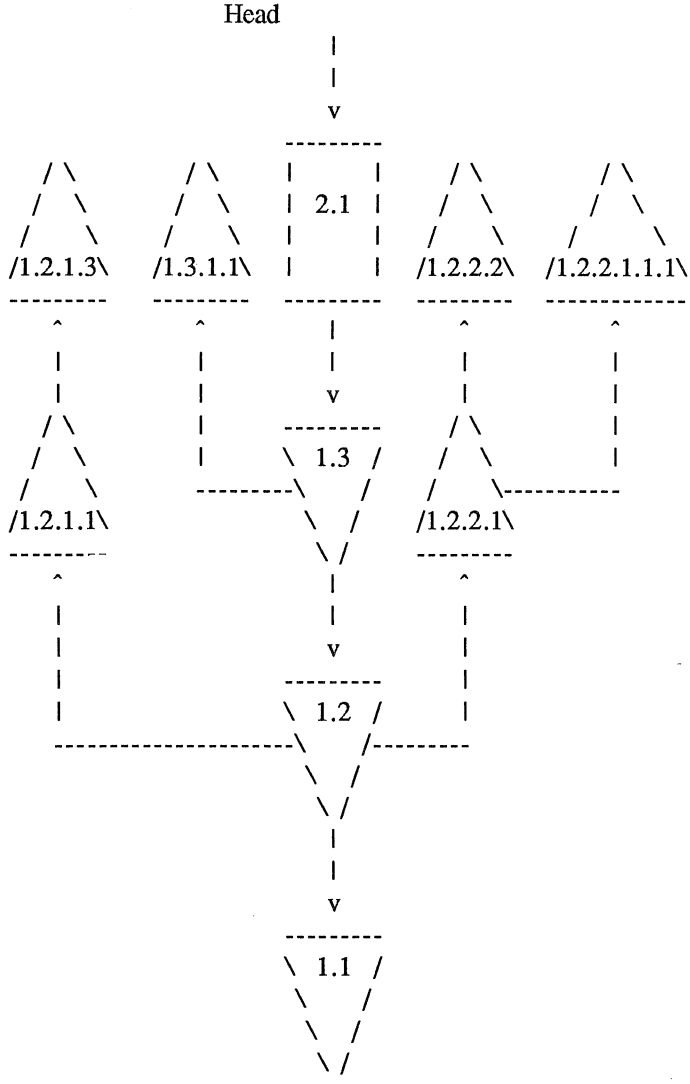

Fig. 1: A revision tree

IDENTIFICATION

Author: Walter F. Tichy, Purdue University, West Lafayette, IN, 47907.
Revision Number: 1.5 ; Release Date: 89/09/12 .

SEE ALSO

ci(1), co(1), ident(1), rcs(1), rcsdiff(1), rcsintro(1), rcsmerge(1), rlog(1).

NAME

reloc – relocation information for a common object file

SYNOPSIS

#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 */
        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 extern symb tab
                                       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, then the reference is a local relocation entry and the *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 reloaction 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 symbols vierual 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 the symbol's 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 the low 16-bits of 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

Assembly Language Programmer's Guide, chapter **Object File Format**, section **Section Relocation Information.**
as(1), ld(1), a.out(4), scnhdr(4), syms(4).

NAME

resolver – host-address resolver configuration file

SYNOPSIS

/usr/etc/resolv.conf

DESCRIPTION

This file controls the behavior of *gethostbyname*(3N), *gethostbyaddr*(3N) and the *resolver*(3N) routines in the C library. It is read by these routines the first time they are invoked by a process.

The file is designed to be human readable and contains a list of keywords with values that provide various types of resolver information. The keyword and value must appear on a single line, and the keyword (e.g., **nameserver**) must start the line. The value follows the keyword, separated by white space.

This file is not necessary if there is a name server running on the local machine and the host name contains the domain name. It is necessary, however, if the system administrator wants to override the default ordering of the host lookup services.

The following configuration option applies to *gethostbyname*(3N) and *gethostbyaddr*(3N):

hostresorder

A list specifying the ordering of host lookup services used by *gethostbyname*(3N) and *gethostbyaddr*(3N). The recognized services and their keywords are Yellow Pages (''yp''), BIND (''bind'') and /etc/hosts (''local''). The keywords are separated by white space or a slash (/). Normally, if a service cannot find the answer or is not running, the next service in the list is queried. The slash separator indicates the previous service in the list is authoritative: even if it cannot find the answer, the search is terminated. For example,

```
hostresorder   bind  local
```

specifies that BIND is checked first (bypassing YP) and if no answer is found, the file /etc/hosts is then checked. At least one service keyword must be listed. The default is ''yp / bind / local'', i.e., YP and BIND have authoritative information if they are available. A user may override the ordering specified by **hostresorder** with the environment variable HOSTRESORDER set to a string containing the service keywords.

The following options are used by the Internet Domain Name System resolver routines only:

nameserver
Internet address (in dot notation) of a name server that the resolver should query. Up to 3 of these lines may be specified; the resolver library queries them in the order listed. If no **nameserver** 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, try the next, until out of name servers, then repeat trying all the name servers until a maximum number of retries are made). When specifying a **nameserver** entry for the local machine, use the address 0 instead of the "localhost" address of 127.1.

domain Local domain name. Most queries for names within this domain can use short names relative to the local domain. If no **domain** entry is present, the domain is determined from the local host name returned by *gethostname* (2); the domain part is taken to be everything after the first '.'. Finally, if the host name does not contain a domain part, the root domain is assumed.

search Search list for host-name lookup. The search list is normally determined from the local domain name; by default, it begins with the local domain name, then successive parent domains that have at least two components in their names. This may be changed by listing the desired domain search path following the *search* keyword with spaces or tabs separating the names. Most resolver queries will be attempted using each component of the search path in turn until a match is found. Note that this process may be slow and will generate a lot of network traffic if the servers for the listed domains are not local, and that queries will time out if no server is available for one of the domains.

The search list is currently limited to six domains with a total of 256 characters. The first item in the list becomes the default domain name; the remaining items are the other domains to search after the default one.

The **domain** and **search** keywords are mutually exclusive. If more than one instance of these keywords is present, the last instance will override. These keywords are ignored if the environment variable LOCALDOMAIN is set.

NOTE

The **hostresorder** keyword is specific to IRIX.

FILES

/usr/etc/resolv.conf

ENVIRONMENT

HOSTRESORDER	overrides **hostresorder**
LOCALDOMAIN	overrides **domain** and **search**

SEE ALSO

gethostbyname(3N), resolver(3N), sethostresorder(3N), hostname(5), named(1M)

Network administration chapters in the *Network Communications Guide*

NAME

rhosts – list of trusted hosts and users

DESCRIPTION

Each user may have a *.rhosts* file in his home directory. This file contains a list of users on other hosts in the network that are trusted in the following sense: when making requests to access the user's system with *rcp*(1C), *rdist*(1C), *rlogin*(1C), or *rsh*(1C), they are allowed to assume the user's identity without specifying a password. In other words, the remote user has exactly the same access privileges on the local system that the owner of the *.rhosts* file does and this access is granted without any attempt to verify the remote user's identity by requiring him to enter a password. The incoming request includes the user name that should be used on the local system. The *.rhosts* file owned by that local user acts as a logical extension to the *hosts.equiv*(4) file when deciding whether to grant permission for the incoming *rcp*(1C), *rdist*(1C), *rlogin*(1C), or *rsh*(1C) request.

The *.rhosts* file has the same format as the *hosts.equiv*(4) file.

NOTES

The owner of the *.rhosts* file must be the super-user (i.e., *root*) or the user in whose home directory it resides. The contents of the file will be disregarded if it is owned by another user or if its permissions allow anyone who is not the owner to modify the file. Use the *chmod*(1) command to add the proper protection:

```
chmod  go-w  .rhosts
```

Special care should be taken in deciding the contents of the file */.rhosts*. Any host or user added to this file has the ability to become the superuser on the local system without entering the password. Note that */.rhosts* is not required.

FILES

~/.rhosts

SEE ALSO

rcp(1C), rdist(1C), rlogin(1C), rsh(1C), hosts.equiv(4)

NAME

rpc – RPC program number data base

SYNOPSIS

/etc/rpc

DESCRIPTION

The *rpc* file contains user readable names that can be used in place of Sun RPC program numbers. Each line has the following information:

name of server for the RPC program
RPC program number
aliases

Items are separated by any number of blanks and/or tab characters. A "#" indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file.

Here is an excerpt of the */etc/rpc* file on IRIX:

```
#
#     rpc   1.10   87/04/10
#
portmapper      100000  portmap sunrpc
rstatd          100001  rstat rup perfmeter
rusersd         100002  rusers
nfs             100003  nfsprog
ypserv          100004  ypprog
mountd          100005  mount showmount
ypbind          100007
walld           100008  rwall shutdown
sgi_toolkitbus  391001
sgi_fam         391002
```

FILES

/etc/rpc

SEE ALSO

getrpcent(3N)

NAME

sccsfile – format of SCCS file

DESCRIPTION

An SCCS (Source Code Control System) file is an ASCII file. It consists of six logical parts: the *checksum*, the *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 the *body* (contains the actual text lines intermixed with control lines).

Throughout an SCCS file there are lines which 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 below which is not depicted as beginning with the control character is prevented from beginning with the control character.

Entries of the form **DDDDD** represent a five-digit string (a number between 00000 and 99999).

Each logical part of an SCCS file is described in detail below.

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 <comments> **...**
.

.
.
@e

The first line (@s) contains the number of lines inserted/deleted/unchanged, respectively. The second line (@d) contains the type of the delta (currently, 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/or numerical group IDs of users who may add deltas to the file, separated by new-lines. The lines containing these login names and/or numerical group IDs are surrounded by the bracketing lines @u and @U. An empty list allows anyone to make a delta. Any line starting with a ! prohibits the succeeding group or user from making deltas.

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 <keyword string>
@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 gotten, or the delta will not be made). When the **b** flag is present the **–b** keyletter may be used on 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 (e.g., 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 SID. The **l** flag defines a *list* of releases that are *locked* against editing (*get*(1) with the **–e** keyletter). The **q** flag defines the replacement for the **%Q%** identification keyword. The **z** flag is used in certain specialized interface programs. *Comments* Arbitrary text is surrounded by the bracketing lines **@t** and **@T**. The comments section typically will contain 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:

@I DDDDD
@D DDDDD
@E DDDDD

respectively. The digit string is the serial number corresponding to the delta for the control line.

SEE ALSO

admin(1), delta(1), get(1), prs(1).

NAME

scnhdr – section header for a MIPS object file

SYNOPSIS

#include <scnhdr.h>

DESCRIPTION

Every MIPS object file has a table of section headers to specify the layout of the data within the file. Each section within an object file has its own header. The C structure appears below:

```
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*(4)]. 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. But it can have no relocation entries or data. Consequently, an uninitialized section has no raw 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*> [*linenum*(4)] 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** so 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 there is no "small" section then the gp table for it is attached to the coresponding "large" section so the information still gets to the link editor, *ld*(1). The C union for the gp table appears below.

```
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 applicable value of the **–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 below 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;          /* data size in bytes, padded to DW boundary */
        long    bsize;          /* bss size in bytes, padded to DW boundary */
        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(4), linenum(4), reloc(4).

BUGS

The *s_nreloc* field has been known to overflow on fully linked objects when the relocation entries are saved.

NAME

scr_dump – format of curses screen image file.

SYNOPSIS

scr_dump(file)

DESCRIPTION

The *curses*(3X) function *scr_dump*() will copy the contents of the screen into a file. The format of the screen image is as described below.

The name of the tty is 20 characters long and the modification time (the *mtime* of the tty that this is an image of) is of the type *time_t*. All other numbers and characters are stored as *chtype* (see **<curses.h>**). No newlines are stored between fields.

```
<magic number:  octal 0433>
<name of tty>
<mod time of tty>
<columns> <lines>
<line length> <chars in line>          for each line on the screen
<line length> <chars in line>
  .
  .
  .
<labels?>                              1, if soft screen labels are present
<cursor row> <cursor column>
```

Only as many characters as are in a line will be listed. For example, if the *<line length>* is **0**, there will be no characters following *<line length>*. If *<labels?>* is TRUE, following it will be

```
<number of labels>
<label width>
<chars in label 1>
<chars in label 2>
  .
  .
  .
```

SEE ALSO

curses(3X).

NAME

services – service name data base

DESCRIPTION

The **/etc/services** file 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

Items are separated by any number of blanks and/or tab characters. The port number and protocol name are considered a single *item*; a "/" is used to separate the port and protocol (e.g. "512/tcp"). A "#" indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file.

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

YELLOW PAGES

If the NFS option is installed and Yellow Pages is running, the *getservent*(3N) library routines do not access this file.

FILES

/etc/services

SEE ALSO

getservent(3N)

BUGS

A name server should be used instead of a static file.

NAME

syms – MIPS symbol table

SYNOPSIS

#include <sym.h>
#include <symconst.h>

DESCRIPTION

The MIPS symbol table departs from the standard COFF symbol table. The symbol table consists of many tables unbundling information usually found in the one COFF symbol table. The symbol table should be viewed as a hand-crafted, 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 info 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 desc.	indirection for inter-file symbol access.
optimzation symbols	to be defined.
auxiliary symbols	variable data type info for each local symbol.
external symbols	loader symbols (global text and data).
external strings	string space for external symbols.
dense numbers	(file, symbol) index pairs 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;    /* addr, size, etc., depends on sc & 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

The chapter on ''The Symbol Table'' in the *Assembly Language Programmers Guide*.
ldfcn(2).

NAME

sys_id – system identification file

DESCRIPTION

The file **/etc/sys_id** contains the name by which the system will be known on communications networks. The name should be no more than eight lower-case letters and digits (to maintain compatibility with foreign networks) and be terminated with a trailing newline. During system startup this file is read by the script **/etc/rc2.d/S20sysetup** and the contents are passed as a parameter to *hostname*(1) to initialize the system name. Once this has been done, this name will returned by the commands *hostname*(1) and *uname*(1) and the system calls *gethostname*(2) and *uname*(2).

FILES

/etc/sys_id

SEE ALSO

hostname(1), uname(1), gethostname(2), uname(2).

NAME

system – system configuration information table

DESCRIPTION

This file is used by the **lboot** program to obtain configuration information. This file generally contains information used to determine if specified hardware exists, a list of software drivers to include in the load, the assignment of system devices such as *pipedev* and *swapdev*, as well as instructions for manually overriding the drivers selected by the self-configuring boot process.

The syntax of the system file is given below. The parser for the **/usr/sysgen/system** file is case sensitive. All upper case strings in the syntax below should be upper case in the **/usr/sysgen/system** file as well. Non-terminal symbols are enclosed in angle brackets "<>" while optional arguments are enclosed in square brackets "[]". Ellipses "..." indicate optional repetition of the argument for that line.

<fname> ::= master file name from */master.d* directory
<func> ::= interrupt function name
<device> ::= special device name | DEV(<major>,<minor>)
<major> ::= <number>
<minor> ::= <number>
<proc> ::= processor # as interpreted by runon(1)
<number> ::= decimal, octal or hex literal

Lboot can determine if hardware exists for a given module by use of *probe* commands. The syntax for probe commands is:

<probe_cmd> ::= probe=<number> [probe_size=<number>]
| <extended_probe>
<extended_probe> ::= exprobe=<probe_sequence>
| exprobe=(<probe_sequence>,<probe_sequence>, ...)
<probe_sequence> ::= (<seq>,<address>,<size>,<value>,<mask>)
<seq> ::= a sequence of 1 or more r's or w's, indicating a read from <address>, or a write to <address>.
<address> ::= <number>
<size> ::= <number>
<value> ::= <number>
<mask> ::= <number>

As shown from the grammar, there are two forms of probe commands. The first allows the specification of an address to read, and optionally, a number of bytes to read. If a probe address is specified, the boot program will attempt to read probe_size bytes (default 4) to determine if the hardware

exists for the module. If the read succeeds, the hardware will be assumed to exist, and the module will be included.

The extended form specifies a sequence of one or more five-tuples used to determine if the hardware exists. Each five-tuple specifies a read/write *sequence*, an *address* to read or write, a *size* of up to four bytes, a *value*, and a *mask*. Then, for each five-tuple, the following is performed:

```
for each element in command do
        if element == 'w' then
                if write(address, value & mask, size) != size then
                        failure
        if element == 'r' then
                if read(address, temp, size) != size then
                        failure
                if temp & mask != value & mask then
                        failure
```

The lines listed below may appear in any order. Blank lines may be inserted at any point. Comment lines must begin with an asterisk. Entries for VECTOR, EXCLUDE and INCLUDE are cumulative. For all other entries, the last line to appear in the file is used -- any earlier entries are ignored.

VECTOR: (Note: this is one line) module=<fname> [intr=<func>]
[vector=<number> ipl=<number> unit=<number>] [base=<number>]
[base2=<number>] [base3=<number>]
[<probe_cmd>]

specifies hardware to conditionally load. If a probe command is specified, the boot program will perform the probe sequence, as discussed above. If the sequence succeeds, the module is included. If a probe sequence is not specified, the hardware will be assumed to exist. The intr function specifies the name of the module's interrupt handler. If it is not specified, the prefix defined in the module's master file (see *master(4)*) is concatenated with the string "intr", and, if a routine with that name is found in the module's object (which resides in the directory **/usr/sysgen/boot**, it is used as the interrupt routine. If the triplet (vector, ipl, unit, base) is specified, a VME interrupt structure is assigned, using the corresponding VME address "vector", priority level "ipl", unit "unit". If the modules' object contains a routine whose name is the concatenation of the master file prefix and ''edtinit'', that routine is involved once at startup and passed a pointer to an edt structure which contains the values for base, base2, base3, and a pointer to

the VME interrupt structure.

EXCLUDE: [<string>] ...
specifies drivers to exclude from the load even if the device is found via VECTOR information.

INCLUDE: [<string>[(<number>)]] ...
specifies software drivers or loadable modules to be included in the load. This is necessary to include the drivers for software "devices". The optional <number> (parenthesis required) specifies the number of "devices" to be controlled by the driver (defaults to 1). This number corresponds to the builtin variable *##c* which may be referred to by expressions in part two of the **/usr/sysgen/master** file.

ROOTDEV: <device>
identifies the device containing the root file system.

SWAPDEV: <device> <number> <number>
identifies the device to be used as swap space, the block number the swap space starts at, and the number of swap blocks available.

PIPEDEV: <device>
identifies the device to be used for pipe space.

DUMPDEV: <device>
identifies the device to be used for kernel dumps.

IPL: <IRQ level> <proc>
send VME interrupt at <IRQ level> to <proc>. If <proc> does not exist at run time, the kernel will default to use processor 0.

NETWORKPROC: <proc>
select <proc> to handle all of kernel's networking activities. If <proc> does not exist at run time, the kernel will default to use processor 0.

USE: [<string>[(<number>)] [<extended_probe>]] ...
If the driver is present, it is the same as INCLUDE. Behaves like EXCLUDE if the module or driver is not present in **/usr/sysgen/boot**.

KERNEL: [<string>] ...
Specifies the module containing the heart of the operating system. It must be present in the system file.

LCOPTS

LDOPTS
are option strings given to *cc*(1) and *ld*(1) respectively, to compile the master.c file and link the operating system.

FILES

/usr/sysgen/system
/usr/include/sys/edt.h

SEE ALSO

master(4).
lboot(1M)

NAME

term – format of compiled term file.

SYNOPSIS

/usr/lib/terminfo/?/*

DESCRIPTION

Compiled *terminfo*(4) descriptions are placed under the directory */usr/lib/terminfo*. In order to avoid a linear search of a huge UNIX system directory, a two-level scheme is used: */usr/lib/terminfo/c/name* where *name* is the name of the terminal, and *c* is the first character of *name*. Thus, **att4425** can be found in the file */usr/lib/terminfo/a/att4425*. Synonyms for the same terminal are implemented by multiple links to the same compiled file.

The format has been chosen so that it will be the same on all hardware. An 8-bit byte is assumed, but no assumptions about byte ordering or sign extension are made. Thus, these binary *terminfo*(4) files can be transported to other hardware with 8-bit bytes.

Short integers are stored in two 8-bit bytes. The first byte contains the least significant 8 bits of the value, and the second byte contains the most significant 8 bits. (Thus, the value represented is 256*second+first.) The value **–1** is represented by **0377,0377**, and the value **–2** is represented by **0376,0377**; other negative values are illegal. Computers where this does not correspond to the hardware read the integers as two bytes and compute the result, making the compiled entries portable between machine types. The **–1** generally means that a capability is missing from this terminal. The **–2** means that the capability has been cancelled in the *terminfo*(4) source and also is to be considered missing.

The compiled file is created from the source file descriptions of the terminals (see the **–I** option of *infocmp*(1M)) by using the *terminfo*(4) compiler, *tic*(1M), and read by the routine **setupterm**(). (See *curses*(3X).) The file is divided into six parts: the header, terminal names, boolean flags, numbers, strings, and string table.

The header section begins the file. This section contains six short integers in the format described below. These integers are (1) the magic number (octal **0432**); (2) the size, in bytes, of the names section; (3) the number of bytes in the boolean section; (4) the number of short integers in the numbers section; (5) the number of offsets (short integers) in the strings section; (6) the size, in bytes, of the string table.

The terminal names section comes next. It contains the first line of the *terminfo*(4) description, listing the various names for the terminal, separated by the bar (|) character (see *term*(5)). The section is terminated with an ASCII NUL character.

The boolean flags have one byte for each flag. This byte is either **0** or **1** as the flag is present or absent. The value of **2** means that the flag has been cancelled. The capabilities are in the same order as the file **<term.h>**.

Between the boolean section and the number section, a null byte will be inserted, if necessary, to ensure that the number section begins on an even byte. All short integers are aligned on a short word boundary.

The numbers section is similar to the boolean flags section. Each capability takes up two bytes, and is stored as a short integer. If the value represented is **–1** or **–2**, the capability is taken to be missing.

The strings section is also similar. Each capability is stored as a short integer, in the format above. A value of **–1** or **–2** means the capability is missing. Otherwise, the value is taken as an offset from the beginning of the string table. Special characters in ^X or \c notation are stored in their interpreted form, not the printing representation. Padding information ($<nn>) and parameter information (%x) are stored intact in uninterpreted form.

The final section is the string table. It contains all the values of string capabilities referenced in the string section. Each string is null terminated.

Note that it is possible for **setupterm**() to expect a different set of capabilities than are actually present in the file. Either the database may have been updated since **setupterm**() has been recompiled (resulting in extra unrecognized entries in the file) or the program may have been recompiled more recently than the database was updated (resulting in missing entries). The routine **setupterm**() must be prepared for both possibilities – this is why the numbers and sizes are included. Also, new capabilities must always be added at the end of the lists of boolean, number, and string capabilities.

Some limitations: total compiled entries cannot exceed 4096 bytes; all entries in the name field cannot exceed 128 bytes.

FILES

/usr/lib/terminfo/?/*	compiled terminal description database
/usr/include/term.h	*terminfo*(4) header file

SEE ALSO

curses(3X), terminfo(4), term(5).
infocmp(1M) in the *System Administrator's Reference Manual*.
Chapter 9 of the *Programmer's Guide*.

NAME

terminfo – terminal capability data base

SYNOPSIS

/usr/lib/terminfo/?/*

DESCRIPTION

terminfo is a compiled database (see *tic*(1M)) describing the capabilities of terminals. Terminals are described in *terminfo* source descriptions by giving a set of capabilities which they have, by describing how operations are performed, by describing padding requirements, and by specifying initialization sequences. This database is used by applications programs, such as *vi*(1) and *curses*(3X), so they can work with a variety of terminals without changes to the programs. To obtain the source description for a terminal, use the **–I** option of *infocmp*(1M).

Entries in *terminfo* source files consist of a number of comma-separated fields. White space after each comma is ignored. The first line of each terminal description in the *terminfo* database gives the name by which *terminfo* knows the terminal, separated by bar (|) characters. The first name given is the most common abbreviation for the terminal (this is the one to use to set the environment variable **TERM** in *$HOME/.profile*; see *profile*(4)), the last name given should be a long name fully identifying the terminal, and all others are understood as synonyms for the terminal name. All names but the last should contain no blanks and must be unique in the first 14 characters; the last name may contain blanks for readability.

Terminal names (except for the last, verbose entry) should be chosen using the following conventions. The particular piece of hardware making up the terminal should have a root name chosen, for example, for the AT&T 4425 terminal, **att4425**. Modes that the hardware can be in, or user preferences, should be indicated by appending a hyphen and an indicator of the mode. See *term*(5) for examples and more information on choosing names and synonyms.

CAPABILITIES

In the table below, the **Variable** is the name by which the C programmer (at the *terminfo* level) accesses the capability. The **Capname** is the short name for this variable used in the text of the database. It is used by a person updating the database and by the *tput*(1) command when asking what the value of the capability is for a particular terminal. The **Termcap Code** is a two-letter code that 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. 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.

All string capabilities listed below may have padding specified, with the exception of those used for input. Input capabilities, listed under the **Strings** section in the table below, have names beginning with **key_**. The following indicators may appear at the end of the **Description** for a variable.

(G) indicates that the string is passed through **tparm**( ) with parameters (parms) as given ($\#_i$).

(*) indicates that padding may be based on the number of lines affected.

($\#_i$) indicates the i^{th} parameter.

Variable	Cap-name	Termcap Code	Description
Booleans			
auto_left_margin	bw	bw	**cub1** wraps from column 0 to last column
auto_right_margin	am	am	Terminal has automatic margins
back_color_erase	bce	be	Screen erased with background color
can_change	ccc	cc	Terminal can re-define existing color
ceol_standout_glitch	xhp	xs	Standout not erased by overwriting (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 (e.g. dialup, switch).
hard_copy	hc	hc	Hardcopy terminal
hard_cursor	chts	HC	Cursor is hard to see.
has_meta_key	km	km	Has a meta key (shift, sets parity bit)
has_status_line	hs	hs	Has extra "status line"
hue_lightness_saturation	hls	hl	Terminal uses only HLS color notation (Tektronix)
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
needs_xon_xoff	nxon	nx	Padding won't work, xon/xoff required
no_esc_ctlc	xsb	xb	Beehive (f1=escape, f2=ctrl C)
non_rev_rmcup	nrrmc	NR	**smcup** does not reverse **rmcup**
no_pad_char	npc	NP	Pad character doesn't exist
over_strike	os	os	Terminal overstrikes on hard-copy terminal
prtr_silent	mc5i	5i	Printer won't echo on screen.
status_line_esc_ok	eslok	es	Escape can be used on the status line
dest_tabs_magic_smso	xt	xt	Destructive tabs, magic **smso** char (t1061)

tilde_glitch	hz	hz	Hazeltine; can't print tildes(~)
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.
label_height	lh	lh	Number of rows in each label
label_width	lw	lw	Number of cols in each label
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 blank chars left by **smso** or **rmso**
max_colors	colors	Co	Maximum number of colors on the screen
max_pairs	pairs	pa	Maximum number of color-pairs on the screen
no_color_video	ncv	NC	Video attributes that can't be used with colors
num_labels	nlab	Nl	Number of labels on screen (start at 1)
padding_baud_rate	pb	pb	Lowest baud rate where padding needed
virtual_terminal	vt	vt	Virtual terminal number (UNIX system)
width_status_line	wsl	ws	Number of columns in status line

Strings

acs_chars	acsc	ac	Graphic charset pairs aAbBcC - def=vt100+
back_tab	cbt	bt	Back tab
bell	bel	bl	Audible signal (bell)
carriage_return	cr	cr	Carriage return (*)
change_scroll_region	csr	cs	Change to lines #1 thru #2 (vt100) (G)
char_padding	rmp	rP	Like **ip** but when in replace mode
clear_all_tabs	tbc	ct	Clear all tab stops
clear_margins	mgc	MC	Clear left and right soft margins
clear_screen	clear	cl	Clear screen and home cursor (*)
clr_bol	el1	cb	Clear to beginning of line, inclusive
clr_eol	el	ce	Clear to end of line
clr_eos	ed	cd	Clear to end of display (*)
column_address	hpa	ch	Horizontal position absolute (G)
command_character	cmdch	CC	Term. settable cmd char in prototype
cursor_address	cup	cm	Cursor motion to row #1 col #2 (G)
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 (G)

cursor_normal	cnorm	ve	Make cursor appear normal (undo **vs/vi**)
cursor_right	cuf1	nd	Non-destructive 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 (*)
delete_line	dl1	dl	Delete line (*)
dis_status_line	dsl	ds	Disable status line
down_half_line	hd	hd	Half-line down (forward 1/2 linefeed)
ena_acs	enacs	eA	Enable alternate char set
enter_alt_charset_mode	smacs	as	Start alternate character set
enter_am_mode	smam	SA	Turn on automatic margins
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 standout mode
enter_underline_mode	smul	us	Start underscore mode
enter_xon_mode	smxon	SX	Turn on xon/xoff handshaking
erase_chars	ech	ec	Erase #1 characters (G)
exit_alt_charset_mode	rmacs	ae	End alternate character set
exit_am_mode	rmam	RA	Turn off automatic margins
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 standout mode
exit_underline_mode	rmul	ue	End underscore mode
exit_xon_mode	rmxon	RX	Turn off xon/xoff handshaking
flash_screen	flash	vb	Visible bell (may not move cursor)
form_feed	ff	ff	Hardcopy terminal page eject (*)
from_status_line	fsl	fs	Return from status line
init_1string	is1	i1	Terminal initialization string
init_2string	is2	is	Terminal initialization string
init_3string	is3	i3	Terminal initialization string
init_file	if	if	Name of initialization file containing **is**

init_prog	iprog	iP	Path name of program for init.
initialize_color	initc	Ic	Initialize the definition of color
initialize_pair	initp	Ip	Initialize color-pair
insert_character	ich1	ic	Insert character
insert_line	il1	al	Add new blank line (*)
insert_padding	ip	ip	Insert pad after character inserted (*)
key_a1	ka1	K1	KEY_A1, 0534, Upper left of keypad
key_a3	ka3	K3	KEY_A3, 0535, Upper right of keypad
key_b2	kb2	K2	KEY_B2, 0536, Center of keypad
key_backspace	kbs	kb	KEY_BACKSPACE, 0407, Sent by backspace key
key_beg	kbeg	@1	KEY_BEG, 0542, Sent by beg(inning) key
key_btab	kcbt	kB	KEY_BTAB, 0541, Sent by back-tab key
key_c1	kc1	K4	KEY_C1, 0537, Lower left of keypad
key_c3	kc3	K5	KEY_C3, 0540, Lower right of keypad
key_cancel	kcan	@2	KEY_CANCEL, 0543, Sent by cancel key
key_catab	ktbc	ka	KEY_CATAB, 0526, Sent by clear-all-tabs key
key_clear	kclr	kC	KEY_CLEAR, 0515, Sent by clear-screen or erase key
key_close	kclo	@3	KEY_CLOSE, 0544, Sent by close key
key_command	kcmd	@4	KEY_COMMAND, 0545, Sent by cmd (command) key
key_copy	kcpy	@5	KEY_COPY, 0546, Sent by copy key
key_create	kcrt	@6	KEY_CREATE, 0547, Sent by create key
key_ctab	kctab	kt	KEY_CTAB, 0525, Sent by clear-tab key
key_dc	kdch1	kD	KEY_DC, 0512, Sent by delete-character key
key_dl	kdl1	kL	KEY_DL, 0510, Sent by delete-line key
key_down	kcud1	kd	KEY_DOWN, 0402, Sent by terminal down-arrow key
key_eic	krmir	kM	KEY_EIC, 0514, Sent by **rmir** or **smir** in insert mode
key_end	kend	@7	KEY_END, 0550, Sent by end key
key_enter	kent	@8	KEY_ENTER, 0527, Sent by enter/send key
key_eol	kel	kE	KEY_EOL, 0517, Sent by clear-to-end-of-line key
key_eos	ked	kS	KEY_EOS, 0516, Sent by clear-to-end-of-screen key
key_exit	kext	@9	KEY_EXIT, 0551, Sent by exit key
key_f0	kf0	k0	KEY_F(0), 0410, Sent by function key f0
key_f1	kf1	k1	KEY_F(1), 0411, Sent by function key f1
key_f2	kf2	k2	KEY_F(2), 0412, Sent by function key f2

key_f3	kf3	k3	KEY_F(3), 0413, Sent by function key f3
key_f4	kf4	k4	KEY_F(4), 0414, Sent by function key f4
key_f5	kf5	k5	KEY_F(5), 0415, Sent by function key f5
key_f6	kf6	k6	KEY_F(6), 0416, Sent by function key f6
key_f7	kf7	k7	KEY_F(7), 0417, Sent by function key f7
key_f8	kf8	k8	KEY_F(8), 0420, Sent by function key f8
key_f9	kf9	k9	KEY_F(9), 0421, Sent by function key f9
key_f10	kf10	k;	KEY_F(10), 0422, Sent by function key f10
key_f11	kf11	F1	KEY_F(11), 0423, Sent by function key f11
key_f12	kf12	F2	KEY_F(12), 0424, Sent by function key f12
key_f13	kf13	F3	KEY_F(13), 0425, Sent by function key f13
key_f14	kf14	F4	KEY_F(14), 0426, Sent by function key f14
key_f15	kf15	F5	KEY_F(15), 0427, Sent by function key f15
key_f16	kf16	F6	KEY_F(16), 0430, Sent by function key f16
key_f17	kf17	F7	KEY_F(17), 0431, Sent by function key f17
key_f18	kf18	F8	KEY_F(18), 0432, Sent by function key f18
key_f19	kf19	F9	KEY_F(19), 0433, Sent by function key f19
key_f20	kf20	FA	KEY_F(20), 0434, Sent by function key f20
key_f21	kf21	FB	KEY_F(21), 0435, Sent by function key f21
key_f22	kf22	FC	KEY_F(22), 0436, Sent by function key f22
key_f23	kf23	FD	KEY_F(23), 0437, Sent by function key f23
key_f24	kf24	FE	KEY_F(24), 0440, Sent by function key f24
key_f25	kf25	FF	KEY_F(25), 0441, Sent by function key f25
key_f26	kf26	FG	KEY_F(26), 0442, Sent by function key f26
key_f27	kf27	FH	KEY_F(27), 0443, Sent by function key f27
key_f28	kf28	FI	KEY_F(28), 0444, Sent by function key f28
key_f29	kf29	FJ	KEY_F(29), 0445, Sent by function key f29
key_f30	kf30	FK	KEY_F(30), 0446, Sent by function key f30
key_f31	kf31	FL	KEY_F(31), 0447, Sent by function key f31
key_f32	kf32	FM	KEY_F(32), 0450, Sent by function key f32
key_f33	kf33	FN	KEY_F(13), 0451, Sent by function key f13
key_f34	kf34	FO	KEY_F(34), 0452, Sent by function key f34
key_f35	kf35	FP	KEY_F(35), 0453, Sent by function key f35
key_f36	kf36	FQ	KEY_F(36), 0454, Sent by function key f36
key_f37	kf37	FR	KEY_F(37), 0455, Sent by function key f37
key_f38	kf38	FS	KEY_F(38), 0456, Sent by function key f38
key_f39	kf39	FT	KEY_F(39), 0457, Sent by function key f39
key_f40	kf40	FU	KEY_F(40), 0460, Sent by function key f40
key_f41	kf41	FV	KEY_F(41), 0461, Sent by function key f41
key_f42	kf42	FW	KEY_F(42), 0462, Sent by function key f42
key_f43	kf43	FX	KEY_F(43), 0463, Sent by function key f43

key_f44	kf44	FY	KEY_F(44), 0464, Sent by function key f44
key_f45	kf45	FZ	KEY_F(45), 0465, Sent by function key f45
key_f46	kf46	Fa	KEY_F(46), 0466, Sent by function key f46
key_f47	kf47	Fb	KEY_F(47), 0467, Sent by function key f47
key_f48	kf48	Fc	KEY_F(48), 0470, Sent by function key f48
key_f49	kf49	Fd	KEY_F(49), 0471, Sent by function key f49
key_f50	kf50	Fe	KEY_F(50), 0472, Sent by function key f50
key_f51	kf51	Ff	KEY_F(51), 0473, Sent by function key f51
key_f52	kf52	Fg	KEY_F(52), 0474, Sent by function key f52
key_f53	kf53	Fh	KEY_F(53), 0475, Sent by function key f53
key_f54	kf54	Fi	KEY_F(54), 0476, Sent by function key f54
key_f55	kf55	Fj	KEY_F(55), 0477, Sent by function key f55
key_f56	kf56	Fk	KEY_F(56), 0500, Sent by function key f56
key_f57	kf57	Fl	KEY_F(57), 0501, Sent by function key f57
key_f58	kf58	Fm	KEY_F(58), 0502, Sent by function key f58
key_f59	kf59	Fn	KEY_F(59), 0503, Sent by function key f59
key_f60	kf60	Fo	KEY_F(60), 0504, Sent by function key f60
key_f61	kf61	Fp	KEY_F(61), 0505, Sent by function key f61
key_f62	kf62	Fq	KEY_F(62), 0506, Sent by function key f62
key_f63	kf63	Fr	KEY_F(63), 0507, Sent by function key f63
key_find	kfnd	@0	KEY_FIND, 0552, Sent by find key
key_help	khlp	%1	KEY_HELP, 0553, Sent by help key
key_home	khome	kh	KEY_HOME, 0406, Sent by home key
key_ic	kich1	kI	KEY_IC, 0513, Sent by ins-char/enter ins-mode key
key_il	kil1	kA	KEY_IL, 0511, Sent by insert-line key
key_left	kcub1	kl	KEY_LEFT, 0404, Sent by terminal left-arrow key
key_ll	kll	kH	KEY_LL, 0533, Sent by home-down key
key_mark	kmrk	%2	KEY_MARK, 0554, Sent by mark key
key_message	kmsg	%3	KEY_MESSAGE, 0555, Sent by message key
key_move	kmov	%4	KEY_MOVE, 0556, Sent by move key
key_next	knxt	%5	KEY_NEXT, 0557, Sent by next-object key
key_npage	knp	kN	KEY_NPAGE, 0522, Sent by next-page key
key_open	kopn	%6	KEY_OPEN, 0560, Sent by open key
key_options	kopt	%7	KEY_OPTIONS, 0561, Sent by options key
key_ppage	kpp	kP	KEY_PPAGE, 0523, Sent by previous-page key
key_previous	kprv	%8	KEY_PREVIOUS, 0562, Sent by previous-object key
key_print	kprt	%9	KEY_PRINT, 0532, Sent by print or copy key
key_redo	krdo	%0	KEY_REDO, 0563, Sent by redo key

key_reference	kref	&1	KEY_REFERENCE, 0564, Sent by ref(erence) key
key_refresh	krfr	&2	KEY_REFRESH, 0565, Sent by refresh key
key_replace	krpl	&3	KEY_REPLACE, 0566, Sent by replace key
key_restart	krst	&4	KEY_RESTART, 0567, Sent by restart key
key_resume	kres	&5	KEY_RESUME, 0570, Sent by resume key
key_right	kcuf1	kr	KEY_RIGHT, 0405, Sent by terminal right-arrow key
key_save	ksav	&6	KEY_SAVE, 0571, Sent by save key
key_sbeg	kBEG	&9	KEY_SBEG, 0572, Sent by shifted beginning key
key_scancel	kCAN	&0	KEY_SCANCEL, 0573, Sent by shifted cancel key
key_scommand	kCMD	*1	KEY_SCOMMAND, 0574, Sent by shifted command key
key_scopy	kCPY	*2	KEY_SCOPY, 0575, Sent by shifted copy key
key_screate	kCRT	*3	KEY_SCREATE, 0576, Sent by shifted create key
key_sdc	kDC	*4	KEY_SDC, 0577, Sent by shifted delete-char key
key_sdl	kDL	*5	KEY_SDL, 0600, Sent by shifted delete-line key
key_select	kslt	*6	KEY_SELECT, 0601, Sent by select key
key_send	kEND	*7	KEY_SEND, 0602, Sent by shifted end key
key_seol	kEOL	*8	KEY_SEOL, 0603, Sent by shifted clear-line key
key_sexit	kEXT	*9	KEY_SEXIT, 0604, Sent by shifted exit key
key_sf	kind	kF	KEY_SF, 0520, Sent by scroll-forward/down key
key_sfind	kFND	*0	KEY_SFIND, 0605, Sent by shifted find key
key_shelp	kHLP	#1	KEY_SHELP, 0606, Sent by shifted help key
key_shome	kHOM	#2	KEY_SHOME, 0607, Sent by shifted home key
key_sic	kIC	#3	KEY_SIC, 0610, Sent by shifted input key
key_sleft	kLFT	#4	KEY_SLEFT, 0611, Sent by shifted left-arrow key
key_smessage	kMSG	%a	KEY_SMESSAGE, 0612, Sent by shifted message key
key_smove	kMOV	%b	KEY_SMOVE, 0613, Sent by shifted move key
key_snext	kNXT	%c	KEY_SNEXT, 0614, Sent by shifted next key
key_soptions	kOPT	%d	KEY_SOPTIONS, 0615, Sent by shifted options key
key_sprevious	kPRV	%e	KEY_SPREVIOUS, 0616, Sent by shifted prev key
key_sprint	kPRT	%f	KEY_SPRINT, 0617, Sent by shifted print key
key_sr	kri	kR	KEY_SR, 0521, Sent by scroll-backward/up key
key_sredo	kRDO	%g	KEY_SREDO, 0620, Sent by shifted redo key
key_sreplace	kRPL	%h	KEY_SREPLACE, 0621, Sent by shifted replace key

key_sright	kRIT	%i	KEY_SRIGHT, 0622, Sent by shifted right-arrow key
key_srsume	kRES	%j	KEY_SRSUME, 0623, Sent by shifted resume key
key_ssave	kSAV	!1	KEY_SSAVE, 0624, Sent by shifted save key
key_ssuspend	kSPD	!2	KEY_SSUSPEND, 0625, Sent by shifted suspend key
key_stab	khts	kT	KEY_STAB, 0524, Sent by set-tab key
key_sundo	kUND	!3	KEY_SUNDO, 0626, Sent by shifted undo key
key_suspend	kspd	&7	KEY_SUSPEND, 0627, Sent by suspend key
key_undo	kund	&8	KEY_UNDO, 0630, Sent by undo key
key_up	kcuu1	ku	KEY_UP, 0403, 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	10	Labels on function key f0 if not f0
lab_f1	lf1	11	Labels on function key f1 if not f1
lab_f2	lf2	12	Labels on function key f2 if not f2
lab_f3	lf3	13	Labels on function key f3 if not f3
lab_f4	lf4	14	Labels on function key f4 if not f4
lab_f5	lf5	15	Labels on function key f5 if not f5
lab_f6	lf6	16	Labels on function key f6 if not f6
lab_f7	lf7	17	Labels on function key f7 if not f7
lab_f8	lf8	18	Labels on function key f8 if not f8
lab_f9	lf9	19	Labels on function key f9 if not f9
lab_f10	lf10	la	Labels on function key f10 if not f10
label_off	rmln	LF	Turn off soft labels
label_on	smln	LO	Turn on soft labels
meta_off	rmm	mo	Turn off "meta mode"
meta_on	smm	mm	Turn on "meta mode" (8th bit)
newline	nel	nw	Newline (behaves like **cr** followed by **lf**)
orig_colors	oc	oc	Set all color(-pair)s to the original ones
orig_pair	op	op	Set default color-pair to the original one
pad_char	pad	pc	Pad character (rather than null)
parm_dch	dch	DC	Delete #1 chars (G*)
parm_delete_line	dl	DL	Delete #1 lines (G*)
parm_down_cursor	cud	DO	Move cursor down #1 lines. (G*)
parm_ich	ich	IC	Insert #1 blank chars (G*)
parm_index	indn	SF	Scroll forward #1 lines. (G)
parm_insert_line	il	AL	Add #1 new blank lines (G*)
parm_left_cursor	cub	LE	Move cursor left #1 spaces (G)
parm_right_cursor	cuf	RI	Move cursor right #1 spaces. (G*)
parm_rindex	rin	SR	Scroll backward #1 lines. (G)

parm_up_cursor	cuu	UP	Move cursor up #1 lines. (G*)
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
plab_norm	pln	pn	Prog label #1 to show string #2
print_screen	mc0	ps	Print contents of the screen
prtr_non	mc5p	pO	Turn on the printer for #1 bytes
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 (G*)
req_for_input	rfi	RF	Send next input char (for ptys)
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 (G)
save_cursor	sc	sc	Save cursor position.
scroll_forward	ind	sf	Scroll text up
scroll_reverse	ri	sr	Scroll text down
set_attributes	sgr	sa	Define the video attributes #1-#9 (G)
set_background	setb	Sb	Set current background color
set_bottom_margin	smgb	Zk	Set bottom margin at current line
set_bottom_margin_parm	smgbp	Zl	Set bottom margin at line #1 or #2 lines from bottom
set_color_pair	scp	sp	Set current color-pair
set_foreground	setf	Sf	Set current foreground color1
set_left_margin	smgl	ML	Set left margin at current line %374%
set_left_margin	smgl	ML	Set left margin at current line %374%
set_left_margin	smgl	ML	Set soft left margin
set_right_margin	smgr	MR	Set soft right margin
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 (G)
tab	ht	ta	Tab to next 8 space hardware tab stop.
to_status_line	tsl	ts	Go to status line, col #1 (G)
underline_char	uc	uc	Underscore one char and move past it
up_half_line	hu	hu	Half-line up (reverse 1/2 linefeed)
xoff_character	xoffc	XF	X-off character
xon_character	xonc	XN	X-on character

SAMPLE ENTRY

The following entry, which describes the *Concept*–100 terminal, is among the more complex entries in the *terminfo* file as of this writing.

```
concept100|c100|concept|c104|c100-4p|concept 100,
        am, db, eo, in, mir, ul, xenl,
        cols#80, lines#24, pb#9600, vt#8,
        bel=^G, blank=\EH, blink=\EC, clear=^L$<2*>,
        cnorm=\Ew, cr=^M$<9>, cub1=^H, cud1=^J,
        cuf1=\E=, cup=\Ea%p1%' '%+%c%p2%' '%+%c,
        cuu1=\E;, cvvis=\EW, dch1=\E^A$<16*>, dim=\EE,
        dl1=\E^B$<3*>, ed=\E^C$<16*>, el=\E^U$<16>,
        flash=\Ek$<20>\EK, ht=\t$<8>, il1=\E^R$<3*>,
        ind=^J, .ind=^J$<9>, ip=$<16*>,
        is2=\EU\Ef\E7\E5\E8\El\ENH\EK\E\0\Eo&\0\Eo\47\E,
        kbs=^h, kcub1=\E>, kcud1=\E<, kcuf1=\E=, kcuu1=\E;,
        kf1=\E5, kf2=\E6, kf3=\E7, khome=\E?,
        prot=\EI, rep=\Er%p1%c%p2%' '%+%c$<.2*>,
        rev=\ED, rmcup=\Ev\s\s\s\s$<6>\Ep\r\n,
        rmir=\E\0, rmkx=\Ex, rmso=\Ed\Ee, rmul=\Eg,
        rmul=\Eg, sgr0=\EN\0, smcup=\EU\Ev\s\s8p\Ep\r,
        smir=\E^P, smkx=\EX, smso=\EE\ED, smul=\EG,
```

Entries may continue onto multiple lines by placing white space at the beginning of each line except the first.
Lines beginning with "#" are taken as comment lines.
Capabilities in
terminfo
are of three types:
boolean capabilities which indicate that the terminal has some particular feature,
numeric capabilities giving the size of the terminal or particular features,
and string capabilities, which give a sequence which can be used to perform particular terminal operations.

Types of Capabilities

All capabilities have names. For instance, the fact that the *Concept* has *automatic margins* (i.e., 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 character '#' and then the value. Thus **cols**, which indicates the number of columns the terminal has, gives the value **80** for the *Concept*. The value may be specified in decimal, octal or hexadecimal using normal C conventions.

Finally, string-valued capabilities, such as **el** (clear to end of line sequence) are given by the two- to five-character capname, an '=', and then a string ending at the next following comma. A delay in milliseconds may appear anywhere in such a capability, enclosed in $<..> brackets, as in

el=\EK$<3>, and padding characters are supplied by **tputs**() (see *curses*(3X)) to provide this delay. The delay can be either a number, e.g., **20**, or a number followed by an '*' (i.e., **3***), a '/' (i.e., **5/**), or both (i.e., **10*/**). A '*' 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 insert character, the factor is still the number of lines affected. This is always one unless the terminal has **in** and the software uses it.) When a '*' 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. (Only one decimal place is allowed.) A '/' indicates that the padding is mandatory. Otherwise, if the terminal has **xon** defined, the padding information is advisory and will only be used for cost estimates or when the terminal is in raw mode. Mandatory padding will be transmitted regardless of the setting of **xon**.

A number of 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**, and **\s** give a newline, linefeed, return, tab, backspace, formfeed, and space, respectively. Other escapes include: **\^** for caret (^); **** for backslash (\); **\,** for comma (,); **\:** for colon (:); and **\0** for null. (**\0** will actually 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 backslash (e.g., \123).

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 example above. Note that capabilities are defined in a left-to-right order and, therefore, a prior definition will override a later definition.

Preparing Descriptions

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**(1) to check that they are correct. Be aware that a very unusual terminal may expose deficiencies in the ability of the **terminfo** file to describe it or the inability of **vi**(1) to work with that terminal. 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 and programs will look there rather than in */usr/lib/terminfo*. To get the padding for insert-line correct (if the terminal manufacturer did not document it) a severe test is to comment out **xon**, edit a large file at 9600 baud with **vi**(1), delete 16 or so lines from the middle of the screen, then hit the **u** key several times quickly. If the display is corrupted, more padding is usually needed. A similar test can be used for insert-character.

Basic Capabilities

The number of columns on each line for the terminal is given by the **cols** numeric capability. If the terminal has a screen, 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 the terminal uses the xon-xoff flow-control protocol, like most terminals, specify **xon**.

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**=\s" because the space would erase the character moved over.

A very important point here is that the local cursor motions encoded in *terminfo* are undefined at the left and top edges of a screen terminal. Programs should never attempt to backspace around the left edge, unless **bw** is given, and should 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 which 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; i.e., **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 screen 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,

while the Lear Siegler ADM–3 is described as

adm3|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)-like escapes (**%x**) in it. 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 in the manner of a Reverse Polish Notation (postfix) calculator. Typically a sequence will push one of the parameters onto the stack and then print it in some format. Often more complex operations are necessary. Binary operations are in postfix form with the operands in the usual order. That is, to get x–5 one would use **%gx%{5}%–**.

The **%** encodings have the following meanings:

%%	outputs '%'
%[[:]*flags*][*width*[.*precision*]][**doxXs**]	as in printf, flags are [–+#] and space
%c	print pop() gives %c
%p[1-9]	push i^{th} parm
%P[a-z]	set variable [a-z] to pop()
%g[a-z]	get variable [a-z] and push it
%'*c*'	push char constant *c*
%{*nn*}	push decimal constant *nn*
%l	push strlen(pop())
%+ %– %* %/ %m	arithmetic (%m is mod): push(pop() op pop())

%& %| %^ bit operations: push(pop() op pop())
%= %> %<logical operations: push(pop() op pop())
%A %O logical operations: and, or
%! %~ unary operations: push(op pop())
%i (for ANSI terminals)
add 1 to first parm, if one parm present,
or first two parms, if more than one parm present

%? 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 b_5%;
c_i are conditions, b_i are bodies.

If the "–" flag is used with "%[doxXs]", then a colon (:) must be placed between the "%" and the "–" to differentiate the flag from the binary "%–" operator, .e.g "%:–16.16s".

Consider the Hewlett-Packard 2645, 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 zero-padded as two digits. Thus its **cup** capability is "**cup**=\E&a%p2%2.2dc%p1%2.2dY$<6>".

The Micro-Term 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 which 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%'\s'%+%c%p2%'\s'%+%c". 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.

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): to the top left corner of the screen, not of memory. (Thus, the \EH sequence on Hewlett-Packard terminals cannot be used for **home** without losing some of the other features on the terminal.)

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 two-parameter sequence (as with the Hewlett-Packard 2645) and can be used in preference to **cup**. If there are parameterized local motions (e.g., 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.

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 beginning of the line to the current position inclusive, leaving the cursor where it is, this should be given as **el1**. If the terminal can clear from the current position to the end of the display, then 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 which the cursor is on, then 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** which take a single parameter and insert or delete that many lines can be given as **il** and **dl**.

If the terminal has a settable destructive 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, alas, 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 is often faster even on terminals with those features.

To determine whether a terminal has destructive scrolling regions or non-destructive scrolling regions, create a scrolling region in the middle of the screen, place data on the bottom line of the scrolling region, move the cursor to the top line of the scrolling region, and do a reverse index (**ri**) followed by a delete line (**dl1**) or index (**ind**). If the data that was originally on the bottom line of the scrolling region was restored into the scrolling region by the **dl1** or **ind**, then the terminal has non-destructive scrolling regions. Otherwise, it has destructive scrolling regions. Do not specify **csr** if the terminal has non-destructive scrolling regions, unless **ind**, **ri**, **indn**, **rin**, **dl**, and **dl1** all simulate destructive scrolling.

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 a full screen may bring non-blank lines up from below or that scrolling back with **ri** may bring down non-blank lines.

Insert/Delete Character

There are two basic kinds of intelligent terminals with respect to insert/delete character operations which 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 which is either eliminated, or expanded to two untyped blanks. You can determine the kind of terminal you have by clearing the screen and then 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, 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, and should give the capability **in**, which stands for "insert null". While these are two logically separate attributes (one line versus multiline insert mode, and special treatment of untyped spaces) we have seen no terminals whose insert mode cannot be described with the single attribute.

terminfo can describe both terminals which have an insert mode and terminals which 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. Now 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 which send a sequence to open a screen position should give it here. (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 padding in **ip** (a string option). Any other sequence which 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, then 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.

If padding is necessary between characters typed while not in insert mode, give this as a number of milliseconds padding in **rmp**.

It is occasionally necessary to move around while in insert mode to delete characters on the same line (e.g., 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 in this case. Omitting **mir** will affect 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* (see *curses*(3X)), representing a good, high contrast, easy-on-the-eyes, format for highlighting error messages and other attention getters. (If you have a choice, reverse-video plus half-bright is good, or reverse-video alone; however, different users have different preferences on different terminals.) 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 Micro-Term 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 a command is necessary before alternate character set mode is entered, give the sequence in **enacs** (enable alternate-character-set mode).

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 non-zero, as the corresponding attribute is on or off. The nine parameters are, in order: standout, underline, reverse, blink, dim, bold, blank, protect, alternate character set. Not all modes need be supported by **sgr**, only those for which corresponding separate attribute commands exist. (See the example at the end of this section.)

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 Hewlett-Packard 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), then this can be given as **flash**; it must not move the cursor. A good flash can be done by changing the screen into reverse video, pad for 200 ms, then return the screen to normal video.

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**. The boolean **chts** should also be given. If there is a way to make the cursor completely invisible, give that as **civis**. The capability **cnorm** should be given which undoes the effects of either 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* 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 the **smcup** sequence will not restore the screen after an **rmcup** sequence is output (to the state prior to outputting **rmcup**), specify **nrrmc**.

If your terminal generates underlined characters by using the underline character (with no special codes needed) even though it does not otherwise overstrike characters, then you should give the capability **ul**. For terminals where a character overstriking another leaves both characters on the screen, give the capability **os**. If overstrikes are erasable with a blank, then this should be indicated by giving **eo**.

Example of highlighting: assume that the terminal under question needs the following escape sequences to turn on various modes.

tparm parameter	attribute	escape sequence
	none	\E[0m
p1	standout	\E[0;4;7m
p2	underline	\E[0;3m
p3	reverse	\E[0;4m
p4	blink	\E[0;5m
p5	dim	\E[0;7m
p6	bold	\E[0;3;4m
p7	invis	\E[0;8m
p8	protect	not available
p9	altcharset	^O (off) ^N(on)

Note that each escape sequence requires a **0** to turn off other modes before turning on its own mode. Also note that, as suggested above, *standout* is set up to be the combination of *reverse* and *dim*. Also, since this terminal has no *bold* mode, *bold* is set up as the combination of *reverse* and *underline*. In addition, to allow combinations, such as *underline+blink*, the sequence to use would be **\E[0;3;5m**. The terminal doesn't have *protect* mode, either, but that cannot be simulated in any way, so **p8** is ignored. The *altcharset* mode is different in that it is either **^O** or **^N** depending on whether it is off or on. If all modes were to be turned on, the sequence would be **\E[0;3;4;5;7;8m^N**.

Now look at when different sequences are output. For example, **;3** is output when either **p2** or **p6** is true, that is, if either *underline* or *bold* modes are turned on. Writing out the above sequences, along with their dependencies, gives the following:

sequence	when to output	terminfo translation
\E[0	always	\E[0
;3	if p2 or p6	%?%p2%p6%|%t;3%;
;4	if p1 or p3 or p6	%?%p1%p3%|%p6%|%t;4%;
;5	if p4	%?%p4%t;5%;
;7	if p1 or p5	%?%p1%p5%|%t;7%;
;8	if p7	%?%p7%t;8%;
m	always	m
^N or ^O	if p9 ^N, else ^O	%?%p9%t^N%e^O%;

Putting this all together into the **sgr** sequence gives:

sgr=\E[0%?%p2%p6%|%t;3%;%?%p1%p3%|%p6%|%t;4%;%?%p5%t;
|5%;%?%p1%p5%%t;7%;%?%p7%t;8%;m%?%p9%t^N%e^O%;,

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 Hewlett-Packard 2621 keys). If the keypad can be set to transmit or not transmit, give these codes as **smkx** and **rmkx**. 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 **kcub1, kcuf1, kcuu1, kcud1,** and **khome** respectively. If there are function keys such as f0, f1, ..., f63, the codes they send can be given as **kf0, kf1, ..., kf63**. If the first 11 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. Further keys are defined above in the capabilities list.

Strings to program function keys can be given as **pfkey, pfloc,** and **pfx**. A string to program their soft-screen labels can be given as **pln**. 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 mode; and **pfx** causes the string to be transmitted to the computer. The capabilities **nlab**, **lw** and **lh** define how many soft labels there are and their width and height. If there are commands to turn the labels on and off, give them in **smln** and **rmln**. **smln** is normally output after one or more **pln** sequences to make sure that the change becomes visible.

Tabs and Initialization

If the terminal has hardware tabs, the command to advance to the next tab stop can be given as **ht** (usually control 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 which 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 **tput init** (see *tput*(1)) 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. 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).

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 must be sent to the terminal each time the user logs in and be output in the following order: run the program **iprog**; output **is1**; output **is2**; set the margins using **mgc**, **smgl** and **smgr**; set the tabs using **tbc** and **hts**; print the file **if**; and finally output **is3**. This is usually done using the **init** option of *tput*(1); see *profile*(4).

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**. Sequences that do a harder reset from a totally unknown state can be given as **rs1**, **rs2**, **rf**, and **rs3**, analogous to **is1**, **is2**, **is3**, and **if**. (The method using files, **if** and **rf**, is used for a few terminals, from */usr/lib/tabset/**; however, the recommended method is to use the initialization and reset strings.) These strings are output by **tput reset**, which is used when the terminal gets into a wedged state. Commands are normally placed in **rs1**, **rs2**, **rs3**, and **rf** only if they produce annoying effects on the screen and are not necessary when logging in. For example, the

command to set a terminal into 80-column mode would normally be part of **is2**, but on some terminals it causes an annoying glitch on the screen and is not normally needed since the terminal is usually already in 80-column mode.

If a more complex sequence is needed to set the tabs than can be described by using **tbc** and **hts**, the sequence can be placed in **is2** or **if**.

If there are commands to set and clear margins, they can be given as **mgc** (clear all margins), **smgl** (set left margin), and **smgr** (set right margin).

Delays

Certain capabilities control padding in the *tty*(7) driver. These are primarily needed by hard-copy terminals, and are used by **tput init** to set tty modes appropriately. Delays embedded in the capabilities **cr**, **ind**, **cub1**, **ff**, and **tab** can be used to set the appropriate delay bits to be set in the tty driver. If **pb** (padding baud rate) is given, these values can be ignored at baud rates below the value of **pb**.

Status Lines

If the terminal has an extra "status line" that is not normally used by software, this fact can be indicated. 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 that go to a given column of the status line and return from the status line can be given as **tsl** and **fsl**. (**fsl** 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 capability **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, the flag **eslok** can be given. A string which 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, e.g., **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**.

Line Graphics

If the terminal has a line drawing alternate character set, the mapping of glyph to character would be given in **acsc**. The definition of this string is based on the alternate character set used in the DEC VT100 terminal, extended slightly with some characters from the AT&T 4410v1 terminal.

glyph name	vt100+ character
arrow pointing right	+
arrow pointing left	,
arrow pointing down	.
solid square block	0
lantern symbol	I
arrow pointing up	–
diamond	‘
checker board (stipple)	a
degree symbol	f
plus/minus	g
board of squares	h
lower right corner	j
upper right corner	k
upper left corner	l
lower left corner	m
plus	n
scan line 1	o
horizontal line	q
scan line 9	s
left tee (├)	t
right tee (┤)	u
bottom tee (┴)	v
top tee (┬)	w
vertical line	x
bullet	~

The best way to describe a new terminal's line graphics set is to add a third column to the above table with the characters for the new terminal that produce the appropriate glyph when the terminal is in the alternate character set mode. For example,

glyph name	vt100+ char	new tty char
upper left corner	l	R
lower left corner	m	F
upper right corner	k	T
lower right corner	j	G
horizontal line	q	,
vertical line	x	.

Now write down the characters left to right, as in "**acsc**=lRmFkTjGq\,x.".

Color Manipulation

Let us define two methods of color manipulation: the Tektronix method and the HP method. The Tektronix method uses a set of N predefined colors (usually 8) from which a user can select "current" foreground and background colors. Thus a terminal can support up to N colors mixed into N*N color-pairs to be displayed on the screen at the same time. When using an HP method the user cannot define the foreground independently of the background, or vice-versa. Instead, the user must define an entire color-pair at once. Up to M color-pairs, made from 2*M different colors, can be defined this way. Most existing color terminals belong to one of these two classes of terminals.

The numeric variables **colors** and **pairs** define the number of colors and color-pairs that can be displayed on the screen at the same time. If a terminal can change the definition of a color (for example, the Tektronix 4100 and 4200 series terminals), this should be specified with **ccc** (can change color). To change the definition of a color (Tektronix method), use **initc** (initialize color). It requires four arguments: color number (ranging from 0 to **colors**–1) and three RGB (red, green, and blue) values (ranging from 0 to 1000).

Tektronix 4100 series terminals use a type of color notation called HLS (Hue Lightness Saturation) instead of RGB color notation. For such terminals one must define a boolean variable **hls**. The last three arguments to the **initc** string would then be HLS values: H, ranging from 0 to 360; and L and S, ranging from 0 to 100.

If a terminal can change the definitions of colors, but uses a color notation different from RGB and HLS, a mapping to either RGB or HLS must be developed.

To set current foreground or background to a given color, use **setf** (set foreground) and **setb** (set background). They require one parameter: the number of the color. To initialize a color-pair (HP method), use **initp** (initialize pair). It requires seven parameters: the number of a color-pair

(range=0 to **pairs**–1), and six RGB values: three for the foreground followed by three for the background. (Each of these groups of three should be in the order RGB.) When **initc** or **initp** are used, RGB or HLS arguments should be in the order "red, green, blue" or "hue, lightness, saturation"), respectively. To make a color-pair current, use **scp** (set color-pair). It takes one parameter, the number of a color-pair.

Some terminals (for example, most color terminal emulators for PCs) erase areas of the screen with current background color. In such cases, **bce** (background color erase) should be defined. The variable **op** (original pair) contains a sequence for setting the foreground and the background colors to what they were at the terminal start-up time. Similarly, **oc** (original colors) contains a control sequence for setting all colors (for the Tektronix method) or color-pairs (for the HP method) to the values they had at the terminal start-up time.

Some color terminals substitute color for video attributes. Such video attributes should not be combined with colors. Information about these video attributes should be packed into the **ncv** (no color video) variable. There is a one-to-one correspondence between the nine least significant bits of that variable and the video attributes. The following table depicts this correspondence.

Attribute	Bit Position	Decimal Value
A_STANDOUT	0	1
A_UNDERLINE	1	2
A_REVERSE	2	4
A_BLINK	3	8
A_DIM	4	16
A_BOLD	5	32
A_INVIS	6	64
A_PROTECT	7	128
A_ALTCHARSET	8	256

When a particular video attribute should not be used with colors, the corresponding **ncv** bit should be set to 1; otherwise it should be set to zero. To determine the information to pack into the **ncv** variable, you must add together the decimal values corresponding to those attributes that cannot coexist with colors. For example, if the terminal uses colors to simulate reverse video (bit number 2 and decimal value 4) and bold (bit number 5 and decimal value 32), the resulting value for **ncv** will be 36 (4 + 32).

Miscellaneous

If the terminal requires other than a null (zero) character as a pad, then this can be given as **pad**. Only the first character of the **pad** string is used. If the terminal does not have a pad character, specify **npc**.

If the terminal can move up or down half a line, this can be indicated 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 control 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) this can be indicated 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)** is the same as **xxxxxxxxxx**.

If the terminal has a settable command character, such as the Tektronix 4025, this can be indicated with **cmdch**. A prototype command character is chosen which is used in all capabilities. This character is given in the **cmdch** capability to identify it. The following convention is supported on some UNIX systems: If the environment variable CC exists, all occurrences of the prototype character are replaced with the character in CC.

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 is one of those supported by the UNIX system virtual terminal protocol, the terminal number can be given as **vt**. A line-turn-around sequence to be transmitted before doing reads should be specified in **rfi**.

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 will not be transmitted. Sequences to turn on and off xon/xoff handshaking may be given in **smxon** and **rmxon**. If the characters used for handshaking are not ^S and ^Q, they may be specified with **xonc** and **xoffc**.

If the terminal has a "meta key" which acts as a shift key, setting the 8th bit of any character transmitted, this fact can be indicated with **km**. Otherwise, software will assume that the 8th bit is parity and it will usually be 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.

Media copy strings which 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 will be sent to the printer. A variation, **mc5p**, takes one parameter, and leaves the printer on for as many characters as the value of the parameter, then turns the printer off. The parameter should not exceed 255. If the text is not displayed on the terminal screen when the printer is on, specify **mc5i** (silent printer). All text, including **mc4**, is transparently passed to the printer while an **mc5p** is in effect.

Special Cases

The working model used by *terminfo* fits most terminals reasonably well. However, some terminals do not completely match that model, requiring special support by *terminfo*. These are not meant to be construed as deficiencies in the terminals; they are just differences between the working model and the actual hardware. They may be unusual devices or, for some reason, do not have all the features of the *terminfo* model implemented.

Terminals which can not display tilde (~) characters, such as certain Hazeltine terminals, should indicate **hz**.

Terminals which ignore a linefeed immediately after an **am** wrap, such as the *Concept* 100, should indicate **xenl**. Those terminals whose cursor remains on the right-most column until another character has been received, rather than wrapping immediately upon receiving the right-most character, such as the VT100, should also indicate **xenl**.

If **el** is required to get rid of standout (instead of writing normal text on top of it), **xhp** should be given.

Those Teleray terminals whose tabs turn all characters moved over to blanks, should indicate **xt** (destructive tabs). This capability is also taken to mean that it is not possible to position the cursor on top of a "magic cookie" therefore, to erase standout mode, it is instead necessary to use delete and insert line.

Those Beehive Superbee terminals which do not transmit the escape or control–C characters, should specify **xsb**, indicating that the f1 key is to be used for escape and the f2 key for control–C.

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 entry

```
att4424-2|Teletype 4424 in display function group ii,
        rev@, sgr@, smul@, use=att4424,
```

defines an AT&T 4424 terminal that does not have the **rev**, **sgr**, and **smul** capabilities, and hence cannot do highlighting. This is useful for different modes for a terminal, or for different user preferences. More than one **use** capability may be given.

FILES

/usr/lib/terminfo/?/*	compiled terminal description database
/usr/lib/.COREterm/?/*	subset of compiled terminal description database
/usr/lib/tabset/*	tab settings for some terminals, in a format appropriate to be output to the terminal (escape sequences that set margins and tabs)

SEE ALSO

curses(3X), printf(3S), term(5).
captoinfo(1M), infocmp(1M), tic(1M), tty(7) in the *System Administrator's Reference Manual*.
tput(1) in the *User's Reference Manual*.
Chapter 9 of the *Programmer's Guide*.

WARNING

As described in the "Tabs and Initialization" section above, a terminal's initialization strings, **is1**, **is2**, and **is3**, if defined, must be output before a *curses*(3X) program is run. An available mechanism for outputting such strings is **tput init** (see *tput*(1) and *profile*(4)).

Tampering with entries in */usr/lib/.COREterm/?/** or */usr/lib/terminfo/?/** (for example, changing or removing an entry) can affect programs such as *vi*(1) that expect the entry to be present and correct. In particular, removing the description for the "dumb" terminal will cause unexpected problems.

NOTE

The *termcap* database (from earlier releases of UNIX System V) may not be supplied in future releases.

NAME

timezone – set default system time zone

SYNOPSIS

/etc/TIMEZONE

DESCRIPTION

This file sets and exports the time zone environmental variable **TZ**.

This file is read by *init*(1) after system boot up and all subsequent processes inherit **TZ** in their environment.

The syntax of **TZ** can be described as follows:

TZ	→	*zone* */ zone signed_time* */ zone signed_time zone* */ zone signed_time zone dst*
zone	→	*letter letter letter*
signed_time	→	*sign time* */ time*
time	→	*hour* */ hour : minute* */ hour : minute : second*
dst	→	*signed_time* */ signed_time ; dst_date , dst_date* */ ; dst_date , dst_date*
dst_date	→	*julian* */ julian / time*
letter	→	*a / A / b / B / ... / z / Z*
hour	→	*00 / 01 / ... / 23*
minute	→	*00 / 01 / ... / 59*
second	→	*00 / 01 / ... / 59*
julian	→	*001 / 002 / .../ 366*
sign	→	*− / +*

EXAMPLES

The contents of **/etc/TIMEZONE** corresponding to the simple example below could be

```
#       Time Zone
TZ=EST5EDT
export TZ
```

A simple setting for New Jersey could be

TZ=EST5EDT

where **EST** is the abbreviation for the main time zone, **5** is the difference, in hours, between GMT (Greenwich Mean Time) and the main time zone, and **EDT** is the abbreviation for the alternate time zone.

The most complex representation of the same setting, for the year 1986, is

TZ="EST5:00:00EDT4:00:00;117/2:00:00,299/2:00:00"

where **EST** is the abbreviation for the main time zone, **5:00:00** is the difference, in hours, minutes, and seconds between GMT and the main time zone, **EDT** is the abbreviation for the alternate time zone, **4:00:00** is the difference, in hours, minutes, and seconds between GMT and the alternate time zone, **117** is the number of the day of the year (Julian day) when the alternate time zone will take effect, **2:00:00** is the number of hours, minutes, and seconds past midnight when the alternate time zone will take effect, **299** is the number of the day of the year when the alternate time zone will end, and **2:00:00** is the number of hours, minutes, and seconds past midnight when the alternate time zone will end.

A southern hemisphere setting such as the Cook Islands could be

TZ="KDT9:30KST10:00;64/5:00,303/20:00"

This setting means that **KDT** is the abbreviation for the main time zone, **KST** is the abbreviation for the alternate time zone, KDT is **9** hours and **30** minutes later than GMT, KST is **10** hours later than GMT, the starting date of KST is the **64**th day at **5** AM, and the ending date of KST is the **303**rd day at **8** PM.

Starting and ending times are relative to the alternate time zone. If the alternate time zone start and end dates and the time are not provided, the days for the United States that year will be used and the time will be 2 AM. If the start and end dates are provided but the time is not provided, the time will be midnight.

NOTES

When the longer format is used, the **TZ** variable must be surrounded by double quotes as shown.

The system administrator must change the Julian start and end days annually if the longer form of the **TZ** variable is used.

Setting the time during the interval of change from the main time zone to the alternate time zone or vice versa can produce unpredictable results.

SEE ALSO

environ(5).
ctime(3C) in the *Programmer's Reference Manual.*

NAME

transferdevice – a shell script specification for extending the WorkSpace menu functions

SYNOPSIS

transferdevice menu
transferdevice versionsOK

DESCRIPTION

transferdevices are shell scripts that implement one or more possible menu actions and are recognized by the standard WorkSpace file typing rules. The *transfermanager* is used to select and customize transfer devices on a per user bases.

Transfer devices must reside in either the directory */etc/transferDevices* or *$HOME/.workspace/locatTransferLinks*, and follow a set of conventions to be recognized as such.

The second line of the transfer device (the line after the shell invocation) must read

```
#transferDevName
```

where **Name** may be any addition to the "transferDev" prefix. If it is desired that a special icon be associated with a device, corresponding FTR and ICON rules must be constructed.

There are two command line arguments that all transfer devices must understand. Both **menu** and **versionsOK** are used by WorkSpace or the Transfer Manager. In response to a menu argument, a transfer device returns a number of lines to stdout. Each line consists of a text token, a space and string of text. Each text token corresponds to an action that the particular transfer device is designed to implement. The text string is used to describe that action.

In answer to the **versionsOK** argument, the device is expected to return (to stdout) either or both of the strings "local" and "remote" (seperated by a space). A response of local indicates that the transfer device may be invoked "as is." A response of "remote" means that if a symbolic link is created from the file *transferdevice.machine* to *transferdevice*, any invocation of *transferdevice.machine* will be understood by the transfer device to mean that the action should be carried out on the remote machine.

FILES

/etc/transferDevice/
~/.workspace/localTransferLinks/

SEE ALSO

transfermanager(1G), cpioDevice(1), rcpDevice(1), tarDevice(1), workspace(1G)
Programming the IRIS WorkSpace

NAME

ttytype – data base of terminal types by port

SYNOPSIS

/etc/ttytype

DESCRIPTION

Ttytype is a database containing, for each tty port on the system, the kind of terminal that is attached to it. There is one line per port, containing the terminal kind (as a name listed in *termcap*(4)), a space, and the name of the tty, minus /dev/.

This information is read by *tset*(1) and by *login*(1) to initialize the TERM environment variable at login time.

EXAMPLE

```
iris-ansi console
iris-ansi systty
vt100     ttyd1
?h19      ttyd2
?h19      ttyd3
?v50am    ttyd4
?v50am    ttyd5
?v50am    ttyd6
?v50am    ttyd7
?v50am    ttyd8
?v50am    ttyd9
?v50am    ttyd10
?v50am    ttyd11
?v50am    ttyd12
```

FILES

/etc/ttytype

SEE ALSO

tset(1), login(1).

NAME

unistd – file header for symbolic constants

SYNOPSIS

#include <unistd.h>

DESCRIPTION

The header file *<unistd.h>* lists the symbolic constants and structures not already defined or declared in some other header file.

```
/* Symbolic constants for the "access" routine: */

#define R_OK     4      /*Test for Read permission */
#define W_OK     2      /*Test for Write permission */
#define X_OK     1      /*Test for eXecute permission */
#define F_OK     0      /*Test for existence of File */

#define F_ULOCK  0      /*Unlock a previously locked region */
#define F_LOCK   1      /*Lock a region for exclusive use */
#define F_TLOCK  2      /*Test and lock a region for exclusive use *
#define F_TEST   3      /*Test a region for other processes locks */

/*Symbolic constants for the "lseek" routine: */

#define SEEK_SET 0      /* Set file pointer to "offset" */
#define SEEK_CUR 1      /* Set file pointer to current plus "offset" */
#define SEEK_END 2      /* Set file pointer to EOF plus "offset" */

/*Pathnames:*/

#define GF_PATH  /etc/group  /*Pathname of the group file */
#define PF_PATH  /etc/passwd /*Pathname of the passwd file */
```

NAME

utmp, wtmp – utmp and wtmp entry formats

SYNOPSIS

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

DESCRIPTION

These files, which hold user and accounting information for such commands as *who*(1), *write*(1), and *login*(1), have the following structure as defined by **<utmp.h>**:

```
#define   UTMP_FILE    "/etc/utmp"
#define   WTMP_FILE    "/etc/wtmp"
#define   ut_name      ut_user

struct utmp {
        char    ut_user[8];         /* User login name */
        char    ut_id[4];           /* /etc/inittab id (usually line #) */
        char    ut_line[12];        /* device name (console, lnxx) */
        short   ut_pid;             /* process id */
        short   ut_type;            /* type of entry */
        struct  exit_status {
          short   e_termination;    /* Process termination status */
          short   e_exit;           /* Process exit status */
        } ut_exit;                  /* The exit status of a process
                                     * marked as DEAD_PROCESS. */
        time_t  ut_time;            /* time entry was made */
};
```

```
/* Definitions for ut_type */
#define EMPTY           0
#define RUN_LVL         1
#define BOOT_TIME       2
#define OLD_TIME        3
#define NEW_TIME        4
#define INIT_PROCESS    5       /* Process spawned by "init" */
#define LOGIN_PROCESS   6       /* A "getty" process waiting for login */
#define USER_PROCESS    7       /* A user process */
#define DEAD_PROCESS    8
#define ACCOUNTING      9
#define UTMAXTYPE       ACCOUNTING  /* Largest legal value of ut_type */

/* Special strings or formats used in the "ut_line" field when */
/* accounting for something other than a process */
/* No string for the ut_line field can be more than 11 chars + */
/* a NULL in length */
#define RUNLVL_MSG  "run-level %c"
#define BOOT_MSG    "system boot"
#define OTIME_MSG   "old time"
#define NTIME_MSG   "new time"
```

FILES

/etc/utmp
/etc/wtmp

SEE ALSO

getut(3C).
login(1), who(1), write(1) in the *User's Reference Manual.*

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. *Uudecode(1C)* will ignore 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 6 characters "begin ". The word *begin* is followed by 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 newline). These consist of a character count, followed by encoded characters, followed by a newline. 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, 6 bits per character. All are offset by a space to make the characters printing. 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 garbage will be 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

uuencode(1C), uudecode(1C), uucp(1C), mail(1)

NAME

visuallogin, noiconlogin – select and control console login program

DESCRIPTION

The configuration flag *visuallogin* selects the type of login program used for the graphics console. If *visuallogin* is *on*, the visual login program *pandora*(1) used for logins. If it is *off*, the standard IRIX *login*(1) program is used.

If the configuration flag *noiconlogin* is *on*, *pandora*(1) displays icons for each user. If it is off, icons are not displayed.

The value of the flags can be set to *on* or *off* using *chkconfig*(1M).

FILES

/etc/config/visuallogin
/etc/config/noiconlogin

SEE ALSO

login(1), pandora(1), chkconfig(1M)

NAME

intro – introduction to miscellany

DESCRIPTION

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

NAME

ascii – map of ASCII character set

DESCRIPTION

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

000	nul	001	soh	002	stx	003	etx	004	eot	005	enq	006	ack	007	bel
010	bs	011	ht	012	nl	013	vt	014	np	015	cr	016	so	017	si
020	dle	021	dc1	022	dc2	023	dc3	024	dc4	025	nak	026	syn	027	etb
030	can	031	em	032	sub	033	esc	034	fs	035	gs	036	rs	037	us
040	sp	041	!	042	"	043	#	044	$	045	%	046	&	047	´
050	(	051	)	052	*	053	+	054	,	055	-	056	.	057	/
060	0	061	1	062	2	063	3	064	4	065	5	066	6	067	7
070	8	071	9	072	:	073	;	074	<	075	=	076	>	077	?
100	@	101	A	102	B	103	C	104	D	105	E	106	F	107	G
110	H	111	I	112	J	113	K	114	L	115	M	116	N	117	O
120	P	121	Q	122	R	123	S	124	T	125	U	126	V	127	W
130	X	131	Y	132	Z	133	[	134	\	135	]	136	^	137	_
140	`	141	a	142	b	143	c	144	d	145	e	146	f	147	g
150	h	151	i	152	j	153	k	154	l	155	m	156	n	157	o
160	p	161	q	162	r	163	s	164	t	165	u	166	v	167	w
170	x	171	y	172	z	173	{	174	|	175	}	176	~	177	del

00	nul	01	soh	02	stx	03	etx	04	eot	05	enq	06	ack	07	bel
08	bs	09	ht	0a	nl	0b	vt	0c	np	0d	cr	0e	so	0f	si
10	dle	11	dc1	12	dc2	13	dc3	14	dc4	15	nak	16	syn	17	etb
18	can	19	em	1a	sub	1b	esc	1c	fs	1d	gs	1e	rs	1f	us
20	sp	21	!	22	"	23	#	24	$	25	%	26	&	27	´
28	(	29	)	2a	*	2b	+	2c	,	2d	-	2e	.	2f	/
30	0	31	1	32	2	33	3	34	4	35	5	36	6	37	7
38	8	39	9	3a	:	3b	;	3c	<	3d	=	3e	>	3f	?
40	@	41	A	42	B	43	C	44	D	45	E	46	F	47	G
48	H	49	I	4a	J	4b	K	4c	L	4d	M	4e	N	4f	O
50	P	51	Q	52	R	53	S	54	T	55	U	56	V	57	W
58	X	59	Y	5a	Z	5b	[	5c	\	5d	]	5e	^	5f	_
60	`	61	a	62	b	63	c	64	d	65	e	66	f	67	g
68	h	69	i	6a	j	6b	k	6c	l	6d	m	6e	n	6f	o
70	p	71	q	72	r	73	s	74	t	75	u	76	v	77	w
78	x	79	y	7a	z	7b	{	7c	|	7d	}	7e	~	7f	del

NAME

charset – description of the standard supported character set

DESCRIPTION

A single 8-bit character set, based on ISO 8859-1, is currently supported. Other character sets may be supported in the future.

ISO 8859-1 is an 8-bit single-byte coded character set. This set, Latin Alphabet #1, contains characters used for general purpose applications in typical office environments in at least the following languages: Danish, Dutch, English, Faeroese, Finnish, French, German, Icelandic,Irish, Italian, Norwegian, Portuguese, Spanish, and Swedish.

(NOTE: please see this man page in the *IRIS-4D Programmer's Reference Manual* for the table of characters.)

The ASCII 7-bit character set is a proper subset of 8859-1 (see ASCII(5)). The characters added by ISO 8859-1 each have the most significant bit of the byte on.

NAME

environ – user environment

DESCRIPTION

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

CFTIME The default format string to be used by the *date*(1) command and the **ascftime**( ) and **cftime**( ) routines (see *ctime*(3C)). If **CFTIME** is not set or is null, the default format string specified in the **/lib/cftime/***LANGUAGE* file (if it exists) is used in its place (see *cftime*(4)).

CHRCLASS A value that corresponds to a file in **/lib/chrclass** containing character classification and conversion information. This information is used by commands (such as *cat*(1), *ed*(1), *sort*(1), etc.) to classify characters as alphabetic, printable, upper case, etc. and to convert characters to upper or lower case.

When a program or command begins execution, the tables containing this information are initialized based on the value of **CHRCLASS**. If **CHRCLASS** is non-existent, null, set to a value for which no file exists in **/lib/chrclass**, or errors occur while reading the file, the ASCII character set is used. During execution, a program or command can change the values in these tables by calling the **setchrclass**() routine. For more detail, see *ctype*(3C).

These tables are created using the *chrtbl*(1M) command.

HOME The name of the user's login directory, set by *login*(1) from the password file (see *passwd*(4)).

LANGUAGE A language for which a printable file by that name exists in **/lib/cftime.** This information is used by commands (such as *date*(1), *ls*(1), *sort*(1), etc.) to print date and time information in the language specified.

If **LANGUAGE** is non-existent, null, set to a value for which no file exists in **/lib/cftime**, or errors occur while reading the file, the last language requested will be used. (If no language has been requested, the language **usa_english** is assumed.) For a description of the content of files in **/lib/cftime**, see *cftime*(4).

PATH The sequence of directory prefixes that *sh*(1), *time*(1), *nice*(1), *nohup*(1), etc., apply in searching for a file known by an incomplete path name. The prefixes are separated by colons (:). *login*(1) sets **PATH=:/usr/sbin:/usr/bsd:/bin:/usr/bin:/usr/bin/X11.** (For more detail, see the "Execution" section of the *sh*(1) manual page.)

TERM The kind of terminal for which output is to be prepared. This information is used by commands, such as *more*(1) or *vi*(1), which may exploit special capabilities of that terminal.

USER The user name (from the password file) of the user which is currently running. It is set by *login*(1), *su*(1), *rlogind*(1M), *newgrp*(1), and *cron*(1).

LOGNAME Synonymous with **USER**.

SHELL The path to the user's shell, set from the password file.

TZ Time zone information. The simplest format is **xxx***n***zzz** where **xxx** is the standard local time zone abbreviation, *n* is the difference in hours from GMT (Greenwich Mean Time), and **zzz** is the abbreviation for an alternate time zone (usually the daylight-saving local time zone), if any; for example,

TZ="EST5EDT"

The most complex format allows you to specify the difference in hours of the alternate time zone from GMT and the starting day and time and ending day and time for using this alternate time zone. For example, in 1985 the complex format corresponding to the above simple example is:

TZ="EST5:00:00EDT4:00:00;118/2:00:00,300/2:00:00"

When the above complex format is used, it must be surrounded by double quotes. For more details, see *ctime*(3C) and *timezone*(4).

Further names may be placed in the environment by the *export* command and "name=value" arguments in *sh*(1), by the *setenv* command in *csh*(1), or by *exec*(2). It is unwise to conflict with certain shell variables that are frequently exported by **.profile** or **.cshrc** files: **MAIL, PS1, PS2, IFS** (see *profile*(4) and *cshrc(4)).*

NOTES

Administrators should note the following: if you attempt to set the current date to one of the dates that the standard and alternate time zones change (for example, the date that daylight time is starting or ending), and you

attempt to set the time to a time in the interval between the end of standard time and the beginning of the alternate time (or the end of the alternate time and the beginning of standard time), the results are unpredictable.

SEE ALSO

chrtbl(1M), cftime(4), passwd(4), profile(4), cshrc(4), timezone(4), in the *System Administrator's Reference Manual*.
exec(2), ctime(3C), ctype(3C) in the *Programmer's Reference Manual*.
cat(1), date(1), ed(1), env(1), ls(1), login(1), nice(1), nohup(1), sh(1), csh(1), sort(1), time(1), vi(1) in the *User's Reference Manual*.

NAME

fcntl – file control options

SYNOPSIS

#include <fcntl.h>

DESCRIPTION

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

```
/* fcntl(2) requests */
#define F_DUPFD 0       /* Duplicate fildes */
#define F_GETFD 1       /* Get fildes flags */
#define F_SETFD 2       /* Set fildes flags */
#define F_GETFL 3       /* Get file flags */
#define F_SETFL         4       /* Set file flags */
#define F_GETLK 5       /* Get file lock */
#define F_SETLK 6       /* Set file lock */
#define F_SETLKW        7       /* Set file lock and wait */
#define F_CHKFL 8       /* Check legality of file flag changes */

/* The following apply to sockets only */
#define F_GETOWN        10      /* Get pid receiving SIGIO, SIGURG */
#define F_SETOWN        11      /* Set pid to receive SIGIO, SIGURG */

/* Flags for F_GETFL and F_SETFL fcntl(2) requests */
#define FNDELAY 0x04    /* Non-blocking I/O */
#define FAPPEND 0x08    /* append (writes guaranteed at the end) */
#define FSYNC           0x10    /* synchronous write option */
#define FRCACH          0x20    /* Used for file and record locking cache */
#define FASYNC          0x40    /* interrupt-driven I/O for sockets */
#define FNONBLK 0x80    /* POSIX Non-blocking I/O */

/* open-only modes */
#define FCREAT          0x100   /* create if nonexistent */
#define FTRUNC          0x200   /* truncate to zero length */
#define FEXCL           0x400   /* error if already created */
#define FNOCTTY 0x800   /* POSIX: don't make this controlling tty */

/* Flag values accessible to open(2) and fcntl(2) */
#define O_RDONLY        0
#define O_WRONLY        1
#define O_RDWR  2
#define O_ACCMODE       0x3             /* mask for above access bits */
#define O_NDELAY        FNDELAY         /* Non-blocking I/O */
```

```
#define O_APPEND        FAPPEND         /* append (writes guaranteed at end) */
#define O_SYNC          FSYNC           /* synchronous write option */
#define O_NONBLK        FNONBLK         /* POSIX Non-blocking I/O */

/* Flag values accessible only to open(2) */
#define O_CREAT FCREAT /* open w/ create (uses 3rd open arg) */
#define O_TRUNC FTRUNC/* open w/ truncation */
#define O_EXCL          FEXCL           /* exclusive open */
#define O_NOCTTY        FNOCTTY         /* don't assign as controlling tty */

/* file segment locking control structure */
struct flock {
        short   l_type;
        short   l_whence;
        long    l_start;
        long    l_len;                  /* if 0 then until EOF */
        short   l_sysid;                /* returned with F_GETLK */
        short   l_pid;                  /* returned with F_GETLK */
}

/* file segment locking types */
#define F_RDLCK 01              /* Read lock */
#define F_WRLCK 02              /* Write lock */
#define F_UNLCK 03              /* Remove locks */
#define FD_CLOEXEC      0x1             /* fcntl 1 in lo bit of arg param */
```

SEE ALSO

fcntl(2), open(2).
4.3BSD (for socket-related options)

NAME

hostname – host name resolution description

DESCRIPTION

Hostnames are domains, where a domain is a hierarchical, dot-separated list of subdomains; for example, the machine monet, in the Berkeley subdomain of the EDU subdomain of the Internet would be represented as

```
monet.Berkeley.EDU
```

(with no trailing dot).

Hostnames are often used with network client and server programs, which must generally translate the name to an address for use. (This function is generally performed by the library routine *gethostbyname*(3N).) Hostnames are resolved by the Internet name resolver in the following fashion.

If the name consists of a single component, i.e., contains no dot, and if the environment variable "HOSTALIASES" is set to the name of a file, that file is searched for a string matching the input hostname. The file should consist of lines made up of two white-space separated strings, the first of which is the hostname alias, and the second of which is the complete hostname to be substituted for that alias. For example, to refer to the host "matisse.painters.org" with the alias "henri", use

```
henri   matisse.painters.org
```

If a case-insensitive match is found between the hostname to be resolved and the first field of a line in the file, the substituted name is looked up with no further processing.

If the input name ends with a trailing dot, the trailing dot is removed, and the remaining name is looked up with no further processing.

If the input name does not end with a trailing dot, it is looked up by searching through a list of domains until a match is found. The default search list includes first the local domain, then its parent domains with at least 2 name components (longest first). For example, in the domain CS.Berkeley.EDU, the name lithium.CChem will be checked first as lithium.CChem.CS.Berkeley.EDU and then as lithium.CChem.Berkeley.EDU. Lithium.CChem.EDU will not be tried, as there is only one component remaining from the local domain. The search path can be changed from the default by the *resolv.conf* system-wide configuration file. See the descriptions of the **search** keyword in *resolver*(4).

SEE ALSO

named(1M), gethostbyname(3N), resolver(3N), hosts(4), resolver(4)

NAME

math – math functions and constants

SYNOPSIS

#include <math.h>

DESCRIPTION

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

It defines the 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	Pi (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).

NAME

regexp – regular expression compile and match routines

SYNOPSIS

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

#include <regexp.h>

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

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

extern char *loc1, *loc2, *locs;

extern int circf, sed, nbra;

DESCRIPTION

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

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

GETC() — Return the value of the next character in the regular expression pattern. Successive calls to GETC() should return successive characters of the regular expression.

PEEKC() — Return the next character in the regular expression. Successive calls to PEEKC() should return the same character [which should also be the next character returned by GETC()].

UNGETC(*c*) — Cause the argument *c* to be returned by the next call to GETC() [and PEEKC()]. No more that one character of pushback is ever needed and this character is guaranteed to be the last character read by

GETC(). The value of the macro UNGETC(*c*) is always ignored.

RETURN(*pointer*) This macro is used on normal exit of the *compile* routine. The value of the argument *pointer* is a pointer to the character after the last character of the compiled regular expression. This is useful to programs which have memory allocation to manage.

ERROR(*val*) This is the abnormal return from the *compile* routine. The argument *val* is an error number (see table below for meanings). This call should never return.

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

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

compile(instring, expbuf, endbuf, eof)

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

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

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

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

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

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

step(string, expbuf)

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

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

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

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

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

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

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

EXAMPLES

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

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

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

SEE ALSO

ed(1), expr(1), grep(1), sed(1) in the *User's Reference Manual.*

NAME

stat – data returned by stat system call

SYNOPSIS

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

DESCRIPTION

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

Structure of the result of stat

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

#define S_IFMT    0170000  /* type of file */
#define S_IFDIR   0040000  /* directory */
#define S_IFCHR   0020000  /* character special */
#define S_IFBLK   0060000  /* block special */
#define S_IFREG   0100000  /* regular */
#define S_IFIFO   0010000  /* fifo */
#define S_IFLNK   0120000  /* symbolic link */
#define S_ISUID   04000    /* set user id on execution */
#define S_ISGID   02000    /* set group id on execution */
#define S_ISVTX   01000    /* directory permissions control */
#define S_IREAD   00400    /* read permission, owner */
#define S_IWRITE  00200    /* write permission, owner */
#define S_IEXEC   00100    /* execute/search permission, owner */
#define S_ENFMT   S_ISGID  /* record locking enforcement flag */
#define S_IRWXU   00700    /* read,write, execute: owner */
```

```
#define S_IRUSR  00400   /* read permission: owner */
#define S_IWUSR  00200   /* write permission: owner */
#define S_IXUSR  00100   /* execute permission: owner */
#define S_IRWXG  00070   /* read, write, execute: group */
#define S_IRGRP  00040   /* read permission: group */
#define S_IWGRP  00020   /* write permission: group */
#define S_IXGRP  00010   /* execute permission: group */
#define S_IRWXO  00007   /* read, write, execute: other */
#define S_IROTH  00004   /* read permission: other */
#define S_IWOTH  00002   /* write permission: other */
#define S_IXOTH  00001   /* execute permission: other */
```

SEE ALSO

stat(2), types(5).

NAME

stdarg – variable argument list

SYNOPSIS

#include <stdarg.h>

void va_start (va_list ap, *ParmN*);

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

void va_end (va_list ap);

DESCRIPTION

This set of macros provides a means of writing portable procedures that accept variable argument lists. Routines having variable argument lists (such as *printf*(3)) that do not use stdarg are inherently nonportable, since different machines use different argument passing conventions. The stdarg facility is similar to *varargs*(5), but is based on the ANSI Standard for C.

A variable argument list contains one or more parameters. The rightmost parameter plays a special role, and is designated *ParmN* in this discussion.

va_list is a type suitable for storing information needed by the macros **va_start**, **va_arg**, and **va_end**. The called function must declare a variable (referred to as *ap*) of type *va_list*, used to access the argument list.

The **va_start (ap,** *ParmN*) macro initializes *ap* for subsequent use by **va_arg** and **va_end**. **va_start** must be called before any use of **va_arg**.

The **va_arg (ap,** *type*) macro will return the next argument in the list pointed to by *ap*. The first invocation of **va_arg** returns the value of the argument after that specified by *ParmN*. Successive invocations return the values of the remaining arguments in succession. *type* is the type to which the expected argument will be converted when passed as an argument. In standard C, arguments that are **char** or **short** should be accessed as **int**, **unsigned char** or **unsigned short** are converted to **unsigned int**, and **float** arguments are converted to **double**. Different types can be mixed, but it is up to the routine to know what type of argument is expected.

va_end (ap) is used to finish up.

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

EXAMPLE

```
#include <stdarg.h>
#define MAXARGS        31

void f1(int nptrs, ...)
{
        va_list ap;
        char *array[MAXARGS];
```

```
        int ptr_no = 0;

        if (nptrs > MAXARGS)
                nptrs = MAXARGS;
        va_start(ap, nptrs);
        while (ptr_no < nptrs)
                (array[ptr_no++] = va_arg(ap, char *));
        va_end(ap);
}
```

SEE ALSO

varargs(5).

BUGS

Due to the procedure calling convention on the MIPS processor, floating-point parameters may be inaccessible via *stdarg* unless they appear *after* a parameter of non-floating-point type. Thus, in the code sequence

```
extern int foo(float,...);

foo(1.0,2.0);
```

the parameter *2.0* may be accessed incorrectly. If the function expected an intervening non-floating-point parameter, such as

```
extern int foo(float,...);

foo(1.0,4,2.0);
```

the second floating-point parameter would be accessible as a *double*. No problem is encountered, of course, if the type of the first argument is not floating-point.

Stdarg cannot be used when passing structures as parameters, as it is impossible to determine their alignment at runtime.

It is up to the calling routine to determine how many arguments there are, since it is not possible to determine this from the stack frame. For example, *execl* passes a 0 to signal the end of the list. *Printf* can tell how many arguments are supposed to be there by the format.

The macros *va_start* and *va_end* may be arbitrarily complex; for example, *va_start* might contain an opening brace, which is closed by a matching brace in *va_end*. Thus, they should only be used where they could be placed within a single complex statement.

NAME

term – conventional names for terminals

DESCRIPTION

These names are used by certain commands (e.g., *man*(1), *tabs*(1), *tput*(1), *vi*(1) and *curses*(3X)) and are maintained as part of the shell environment in the environment variable **TERM** (see *sh*(1), *profile*(4), and *environ*(5)).

Entries in *terminfo*(4) source files consist of a number of comma-separated fields. (To obtain the source description for a terminal, use the **–I** option of *infocmp*(1M).) White space after each comma is ignored. The first line of each terminal description in the *terminfo*(4) database gives the names by which *terminfo*(4) knows the terminal, separated by bar (|) characters. The first name given is the most common abbreviation for the terminal (this is the one to use to set the environment variable **TERMINFO** in *$HOME/.profile*; see *profile*(4)), the last name given should be a long name fully identifying the terminal, and all others are understood as synonyms for the terminal name. All names but the last should contain no blanks and must be unique in the first 14 characters; the last name may contain blanks for readability.

Terminal names (except for the last, verbose entry) should be chosen using the following conventions. The particular piece of hardware making up the terminal should have a root name chosen, for example, for the AT&T 4425 terminal, **att4425**. This name should not contain hyphens, except that synonyms may be chosen that do not conflict with other names. Up to 8 characters, chosen from [a–z0–9], make up a basic terminal name. Names should generally be based on original vendors, rather than local distributors. A terminal acquired from one vendor should not have more than one distinct basic name. Terminal sub-models, operational modes that the hardware can be in, or user preferences, should be indicated by appending a hyphen and an indicator of the mode. Thus, an AT&T 4425 terminal in 132 column mode would be **att4425–w**. The following suffixes should be used where possible:

Suffix	Meaning	Example
–w	Wide mode (more than 80 columns)	att4425–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
–np	Number of pages of memory	c100–4p
–rv	Reverse video	att4415–rv

To avoid conflicts with the naming conventions used in describing the different modes of a terminal (e.g., –w), it is recommended that a terminal's root name not contain hyphens. Further, it is good practice to make all terminal names used in the *terminfo*(4) database unique. Terminal entries that are present only for inclusion in other entries via the **use=** facilities should have a '+' in their name, as in **4415+nl.**

Some of the known terminal names may include the following (for a complete list, type: **ls -C /usr/lib/terminfo/?**):

2621,hp2621	Hewlett-Packard 2621 series
2631	Hewlett-Packard 2631 line printer
2631–c	Hewlett-Packard 2631 line printer - compressed mode
2631–e	Hewlett-Packard 2631 line printer - expanded mode
2640,hp2640	Hewlett-Packard 2640 series
2645,hp2645	Hewlett-Packard 2645 series
3270	IBM Model 3270
33,tty33	AT&T Teletype Model 33 KSR
35,tty35	AT&T Teletype Model 35 KSR
37,tty37	AT&T Teletype Model 37 KSR
4000a	Trendata 4000a
4014,tek4014	TEKTRONIX 4014
40,tty40	AT&T Teletype Dataspeed 40/2
43,tty43	AT&T Teletype Model 43 KSR
4410,5410	AT&T 4410/5410 terminal in 80-column mode - version 2
4410–nfk,5410–nfk	AT&T 4410/5410 without function keys - version 1
4410–nsl,5410–nsl	AT&T 4410/5410 without pln defined
4410–w,5410–w	AT&T 4410/5410 in 132-column mode
4410v1,5410v1	AT&T 4410/5410 terminal in 80-column mode - version 1
4410v1–w,5410v1–w	AT&T 4410/5410 terminal in 132-column mode - version 1
4415,5420	AT&T 4415/5420 in 80-column mode
4415–nl,5420–nl	AT&T 4415/5420 without changing labels
4415–rv,5420–rv	AT&T 4415/5420 80 columns in reverse video
4415–rv–nl,5420–rv–nl	AT&T 4415/5420 reverse video without changing labels
4415–w,5420–w	AT&T 4415/5420 in 132-column mode
4415–w–nl,5420–w–nl	AT&T 4415/5420 in 132-column mode

	without changing labels
4415–w–rv,5420–w–rv	AT&T 4415/5420 132 columns in reverse video
4415–w–rv–nl,5420–w–rv–nl	AT&T 4415/5420 132 columns reverse video without changing labels
4418,5418	AT&T 5418 in 80-column mode
4418–w,5418–w	AT&T 5418 in 132-column mode
4420	AT&T Teletype Model 4420
4424	AT&T Teletype Model 4424
4424-2	AT&T Teletype Model 4424 in display function group ii
4425,5425	AT&T 4425/5425
4425–fk,5425–fk	AT&T 4425/5425 without function keys
4425–nl,5425–nl	AT&T 4425/5425 without changing labels in 80-column mode
4425–w,5425–w	AT&T 4425/5425 in 132-column mode
4425–w–fk,5425–w–fk	AT&T 4425/5425 without function keys in 132-column mode
4425–nl–w,5425–nl–w	AT&T 4425/5425 without changing labels in 132-column mode
4426	AT&T Teletype Model 4426S
450	DASI 450 (same as Diablo 1620)
450–12	DASI 450 in 12-pitch mode
500,att500	AT&T-IS 500 terminal
510,510a	AT&T 510/510a in 80-column mode
513bct,att513	AT&T 513 bct terminal
5320	AT&T 5320 hardcopy terminal
5420_2	AT&T 5420 model 2 in 80-column mode
5420_2–w	AT&T 5420 model 2 in 132-column mode
5620,dmd	AT&T 5620 terminal 88 columns
5620–24,dmd–24	AT&T Teletype Model DMD 5620 in a 24x80 layer
5620–34,dmd–34	AT&T Teletype Model DMD 5620 in a 34x80 layer
610,610bct	AT&T 610 bct terminal in 80-column mode
610–w,610bct–w	AT&T 610 bct terminal in 132-column mode
7300,pc7300,unix_pc	AT&T UNIX PC Model 7300
735,ti	Texas Instruments TI735 and TI725
745	Texas Instruments TI745
dumb	generic name for terminals that lack reverse line-feed and other special escape sequences
hp	Hewlett-Packard (same as 2645)
iris–ansi	SGI *wsh*(1) ANSI emulator (40 lines)
iris–ansi–24	SGI *wsh*(1) ANSI emulator (24 lines)

iris–ansi–66	SGI *wsh*(1) ANSI emulator (66 lines)
iris–ansi–net	SGI remote login from *wsh*(1) window
lp	generic name for a line printer
pt505	AT&T Personal Terminal 505 (22 lines)
pt505–24	AT&T Personal Terminal 505 (24-line mode)
rwsiris	SGI remote login from visual 50 emulator
sync	generic name for synchronous Teletype Model 4540-compatible terminals
wsiris	SGI visual 50 terminal emulator

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

FILES

/usr/lib/terminfo/?/* compiled terminal description database

SEE ALSO

curses(3X), profile(4), terminfo(4), environ(5).
man(1), sh(1), stty(1), tabs(1), tput(1), tplot(1G), vi(1) in the *User's Reference Manual.*
infocmp(1M) in the *System Administrator's Reference Manual.*
Chapter 9 of the *Programmer's Guide.*

NOTES

Not all programs follow the above naming conventions.

NAME

types – primitive system data types

SYNOPSIS

#include <sys/types.h>

DESCRIPTION

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

```
typedef  struct { int r[1]; } *physadr;
typedef  long               daddr_t;
typedef  char *             caddr_t;
typedef  unsigned char      unchar;
typedef  unsigned short     ushort;
typedef  unsigned int       uint;
typedef  unsigned long      ulong;
typedef  ulong              ino_t;
typedef  short              cnt_t;
typedef  long               time_t;
typedef  int                label_t[12];
typedef  short              dev_t;
typedef  long               off_t;
typedef  unsigned long      paddr_t;
typedef  int                key_t;
typedef  unsigned char      use_t;
typedef  short              sysid_t;
typedef  short              index_t;
typedef  unsigned int       lock_t;
typedef  unsigned int       size_t;
typedef  unsigned char      u_char;
typedef  unsigned short     u_short;
typedef  unsigned int       u_int;
typedef  unsigned long      u_long;
```

The form *daddr_t* is used for disk addresses except in an i-node on disk, see *fs*(4). Times are encoded in seconds since 00:00:00 GMT, January 1, 1970. The major and minor parts of a device code specify kind and unit number of a device 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

fs(4).

NAME

values – machine-dependent values

SYNOPSIS

#include <values.h>

DESCRIPTION

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

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

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

MINDOUBLE, LN_MINDOUBLE — The minimum positive value of a double-precision floating-point number, and its natural logarithm.

FSIGNIF — The number of significant bits in the mantissa of a single-precision floating-point number.

DSIGNIF — The number of significant bits in the mantissa of a double-precision floating-point number.

SEE ALSO

intro(3), math(5).

NAME

varargs – variable argument list

SYNOPSIS

#include <varargs.h>

function(**va_alist**)
va_dcl
va_list *pvar*;
va_start(*pvar*);
f = **va_arg**(*pvar*, *type*);
va_end(*pvar*);

DESCRIPTION

This set of macros provides a means of writing portable procedures that accept variable argument lists. Routines having variable argument lists (such as *printf*(3)) that do not use varargs are inherently nonportable, since different machines use different argument passing conventions.

va_alist is used in a function header to declare a variable argument list.

va_dcl is a declaration for **va_alist**. Note that there is no semicolon after **va_dcl.**

va_list is a type which can be used for the variable *pvar*, which is used to traverse the list. One such variable must always be declared.

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

va_arg(*pvar*, *type*) will return the next argument in the list pointed to by *pvar*. *Type* is the type to which the expected argument will be converted when passed as an argument. In standard C, arguments that are **char** or **short** should be accessed as **int**, **unsigned char** or **unsigned short** are converted to **unsigned int**, and **float** arguments are converted to **double**. Different types can be mixed, but it is up to the routine to know what type of argument is expected, since it cannot be determined at runtime.

va_end(*pvar*) is used to finish up.

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

EXAMPLE

```
#include <varargs.h>
execl(va_alist)
va_dcl
{
        va_list ap;
        char *file;
        char *args[100];
        int argno = 0;
```

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

SEE ALSO

stdarg(5)

BUGS

Due to the procedure calling convention on the MIPS processor, floating-point parameters may be inaccessible via *varargs* unless they appear *after* a parameter of non-floating-point type. Thus, in the code sequence

```
extern int foo(float,...);

foo(1.0,2.0);
```

the parameter *2.0* may be accessed incorrectly. If the function expected an intervening non-floating-point parameter, such as

```
extern int foo(float,...);

foo(1.0,4,2.0);
```

the second floating-point parameter would be accessible as a *double*.

Varargs cannot be used when passing structures as parameters, as it is impossible to determine their alignment at runtime.

It is up to the calling routine to determine how many arguments there are, since it is not possible to determine this from the stack frame. For example, *execl* passes a 0 to signal the end of the list. *Printf* can tell how many arguments are supposed to be there by the format.

The macros *va_start* and *va_end* may be arbitrarily complex; for example, *va_start* might contain an opening brace, which is closed by a matching brace in *va_end*. Thus, they should only be used where they could be placed within a single complex statement.

NAME

winicons – stowed window image mechanism

DESCRIPTION

When an active window is stowed by the user, an RGB image file may be used to paint the canvas of the stowed window icon. A window icon file must contain the suffix *.icon*.

Window icons are assigned to stowed windows by matching the name that appears in the program's call to the Graphics Library subroutine, `winopen`. Thus, an icon for the GL program *cedit* would have this name:

```
cedit.icon
```

WINICON SEARCH PATH

A directory of default window icons exists in *$NEWSHOME/icons*. You may add or customize window icons by placing them in *$HOME/.4sight/icons*. To find the appropriate window icon for a stowed window, 4Sight first searches *$HOME/.4sight/icons* for a name match. If it is unsuccessful, it seaches the default window icon directory (*$NEWSHOME/icons*). If this is unsuccessful, it uses the prededined icon *$NEWSHOME/icons/default.icon*. If this icon is missing for some reason, 4Sight draws the icon without an image.

CREATING A WINICON FILE

A window icon can be created from any image that can be displayed on the IRIS screen (provided that the tools described below are accessible when the image is displayed).

The following passage describes one possible way to create an icon image file. First, display the image you wish to use with the image tools *ipaste* or *showci*, or simply open a window containing a program from which you want to take an image (make sure that the image is still). Then invoke the image tool *icut* from the command line. *icut* takes a filename as an argument; the image cut from the screen is written to that file.

```
icut foo
```

Place the small rectagular *icut* window away from the image you wish to cut. Place the mouse cursor inside the icut window and hold down the <shift> key. While holding down the key, move the mouse cursor to the upper left corner of the area you wish to cut. Hold down the left mouse button while you move to the lower right corner of the area you wish to cut (the area is not shown on the screen), and when you are ready, let go of the left button. The image is then written to the file.

Note: The image you cut must be scaled to fit the stowed window canvas, so you should attempt to cut an area of the same general shape as the icon. The ratio of window icon width to height is 64:50.

You can use the file obtained with *icut* as a window icon file, and 4Sight will scale and dither it on the fly. However, to increase efficiency and image quality, you may want to scale it yourself. To do so, first run the image tool *istat* on the *icut* file:

```
istat foo
```

istat gives a readout of various image statistics; the important ones for scaling are the first two values, the image width (*xsize*) and image height (*ysize*). The dimension of stowed windows is 50 NeWS (4Sight) points high by 64 NeWS points wide. The ratio of screen pixels to points is 4:3; this yields the following scaling factors:

xscale = 85.33/(xsize)
yscale = 66.67/(ysize)

To scale the image, use the image tool *izoom. izoom* takes in input file, an output file, and width and height scaling factors as arguments. To scale an image file *foo* whose dimensions are 620 pixels wide by 500 pixels high and make it into a console window icon, you would type the following on the command line:

```
izoom foo $HOME/.4sight/icons/console.icon .137 .133
```

NOTES

Some programs do not use this mechanism to draw their window icons; specifically, those that draw their own icons rather than let 4Sight do it for them. Certain NeWS programs, such as the Calculator draw their own icons in PostScript; Graphics Library programs using `iconsize` to draw their icons use Graphics Library commands to do so. Both of these methods override the window icon mechanism described here.

SEE ALSO

ipaste(1G), iconsize(3G)

PERMUTED INDEX

generator; routines for changing generators INITGROUPS(3). /number random,
dsopen, dsclose: communicate with generic SCSI devices. . . . . dslib(3)
ds: generic (user mode) SCSI driver. . . ds(7M)
character or word from a stream. getc, getchar, fgetc, getw: get . . . . getc(3S)
or word from a stream. getc, getchar, fgetc, getw: get character . . getc(3S)
working directory. getcwd: get path-name of current . . getcwd(3C)
put in a file system independent/ getdents: read directory entries and . . getdents(2)
get/set name of current domain. getdomainname, setdomainname: . . getdomainname(2)
4.3bsd software signal facilities GETDTABLESIZE(3). . . . . sigvec:
user, real/ getuid, geteuid, getgid, getegid: get real user, effective . . . getuid(2)
name. getenv: return value for environment . getenv(3C)
user, effective user, real/ getuid, geteuid, getgid, getegid: get real . . . getuid(2)
get_fpc_irr, get_fpc_eir,/ fpc, get_fpc_csr, set_fpc_csr, ... fpc(3C)
swapINX:/ /set_fpc_csr, get_fpc_irr, get_fpc_eir, set_fpc_led, swapRM, . . fpc(3C)
fpc, get_fpc_csr, set_fpc_csr, get_fpc_irr, get_fpc_eir,/ . . . . . fpc(3C)
effective user,/ getuid, geteuid, getgid, getegid: get real user, ... getuid(2)
setgrent, endgrent, fgetgrent: get/ getgrent, getgrgid, getgrnam, ... getgrent(3C)
endgrent, fgetgrent: get/ getgrent, getgrgid, getgrnam, setgrent, ... getgrent(3C)
fgetgrent: get/ getgrent, getgrgid, getgrnam, setgrent, endgrent, ... getgrent(3C)
getgroups: get group access list. . . . getgroups(2)
initialize group access list GETGROUPS(3B). . . . . initgroups:
sethostent,/ gethostbyname, gethostbyaddr, gethostent, ... gethostbyname(3N)
gethostent, sethostent, endhostent,/ gethostbyname, gethostbyaddr, . . . gethostbyname(3N)
gethostbyname, gethostbyaddr, gethostent, sethostent, endhostent,/ . . gethostbyname(3N)
identifier of current host. gethostid, sethostid: get/set unique . . gethostid(2)
name of current host. gethostname, sethostname: get/set . . gethostname(2)
host machine swap_*() - swap the/ gethostsex: get the byte sex of the . . sex(3X)
scaninvent: get hardware inventory/ getinvent, setinvent, endinvent, . . . getinvent(3)
of interval timer. getitimer, setitimer: get/set value . . . getitimer(2)
getlogin: get login name. . . . . getlogin(3C)
termios *termios_p, speed_t speed); GETLVENT(3C). cfsetispeed (struct . int
stream. getmsg: get next message off a . . . getmsg(2)
setnetent, endnetent:/ getnetent, getnetbyaddr, getnetbyname, ... getnetent(3N)
get/ getnetent, getnetbyaddr, getnetbyname, setnetent, endnetent: . getnetent(3N)
getnetbyname, setnetent, endnetent:/ getnetent, getnetbyaddr, ... getnetent(3N)
argument vector. getopt: get option letter from . . . . getopt(3C)
getopt: parse command options. . . . getopt(1)
getopts, getoptcvt: parse command options. . . getopts(1)
options. getopts, getoptcvt: parse command . . getopts(1)
getpagesize: get system page size. . . getpagesize(2)
getpass: read a password. . . . . getpass(3C)
peer. getpeername: get name of connected . getpeername(2)
process group, and parent/ getpid, getpgrp, getppid: get process, ... getpid(2)
process, process group, and parent/ getpid, getpgrp, getppid: get . . . . getpid(2)
and parent process/ getpid, getpgrp, getppid: get process, process group, . getpid(2)
program scheduling priority. getpriority, setpriority: get/set . . . . getpriority(2)
getprotoent, getprotobynumber, getprotobyname, setprotoent,/ . . . . getprotoent(3N)
setprotoent,/ getprotoent, getprotobynumber, getprotobyname, . getprotoent(3N)
getprotobyname, setprotoent,/ getprotoent, getprotobynumber, . . . getprotoent(3N)
getpw: get name from UID. . . . . getpw(3C)
setpwent, endpwent, fgetpwent: get/ getpwent, getpwuid, getpwnam, . . . getpwent(3C)

external data representation (xdr) library routines M_FORK(3P). . xdr:
get rpc port number XDR(3R). . . . . getrpcport:
xedit: simple text editor for X. . . . . xedit(1)
xev: print contents of X events. . . . xev(1)
xeyes: watch over your shoulder. . . xeyes(1)
xfd: font displayer for X. . . . . xfd(1)
Generation of figures under X11. xfig: Facility for Interactive . . . . . xfig(1)
for selecting X11 font names. xfontsel: point & click interface . . . xfontsel(1)
xgc: X graphics demo. . . . . xgc(1)
for X. xhost: server access control program . xhost(1)
xinit: X Window System initializer. . xinit(1)
resource. xkill: kill a client by its X . . . . . xkill(1)
xload: load average display for X. . . xload(1)
xlogo: X Window System logo. . . . xlogo(1)
defined on server. xlsatoms: list interned atoms . . . . xlsatoms(1)
running on a display. xlsclients: list client applications . . . xlsclients(1)
for X. xlsfonts: server font list displayer . . xlsfonts(1)
displayer for X. xlswins: server window list . . . . . xlswins(1)
xmag: magnify parts of the screen. . . xmag(1)
for the X Window System.. xman: Manual page display program . xman(1)
display program.. xmessage: X window system message xmessage(1)
handling system. xmh: X interface to the MH message . xmh(1)
keymaps in X. xmodmap: utility for modifying . . . xmodmap(1)
controller. xmt: Xylogics 1/2 inch magnetic tape . xmt(7M)
xpr: print an X window dump. . . . . xpr(1)
xprop: property displayer for X. . . . xprop(1)
utility. xrdb: X server resource database . . . xrdb(1)
X screen. xrefresh: refresh all or part of an . . . xrefresh(1)
xscope: X Window Protocol Viewer. . xscope(1)
xset: user preference utility for X. . . xset(1)
setting utility for X. xsetroot: root window parameter . . . xsetroot(1)
Window System. Xsgi: SGI Iris server for the X . . . . xsgi(1)
xshowcmap: show colormap. . . . . xshowcmap(1)
a NeWS client. xstart: start up the sgi X server as . . xstart(1)
utility. xstdcmap: X standard colormap . . . xcmap(1)
programs to implement shared/ xstr: extract strings from C . . . . . xstr(1)
xterm: terminal emulator for X. . . . xterm(1)
xwd: dump an image of an X window. xwd(1)
for X. xwininfo: window information utility . xwininfo(1)
xwud: image displayer for X. . . . . xwud(1)
controllers and driver.. xyl, xyl754: Xylogics disk . . . . . xyl(7M)
and driver.. xyl, xyl754: Xylogics disk controllers . . xyl(7M)
controllers and driver.. ipi, xylipi: Xylogics IPI disk . . . . . ipi(7M)
controller. xmt: Xylogics 1/2 inch magnetic tape . . . xmt(7M)
driver.. xyl, xyl754: Xylogics disk controllers and . . . . xyl(7M)
driver.. ipi, xylipi: Xylogics IPI disk controllers and . . ipi(7M)
j0, j1, jn, y0, y1, yn: bessel functions. . . . . bessel(3M)
j0, j1, jn, y0, y1, yn: bessel functions. . . . . bessel(3M)
yacc: yet another compiler-compiler. . yacc(1)
sginap: timed sleep and processor yield function. . . . . sginap(2)
j0, j1, jn, y0, y1, yn: bessel functions. . . . . bessel(3M)