```
 ?E
 MISSING DATA [FOR KEYWORD <x>][; <opt>]

e_miskw(x,opt)
 ?E
 MISSING KEYWORD <x>[; <opt>]

e_ng(s1,s2,...,0)
 NG
 <s1> <s2> ....

e_ofc(x,opt)
 ?E
 INVALID OFC: <x>[; <opt>]

e_ok()
 OK

e_perm(x,opt)
 ?E
 USE OF <x> NOT PERMITTED [; <opt>]

e_pf()
 PF

e_punct(x,y,opt)
 ?E
 INVALID PUNCTUATION; '<x>' SHOULD BE '<y>' [; <opt>]

e_rgerr(x,opt)
 ?E
 RANGE ERROR IN <x>[; <opt>]

e_rlbsy(opt)
 RL
 PROGRAM BUSY[; <opt>]

e_rlovld(opt)
 RL
 SYSTEM OVLD[; <opt>]

e_sched()
 NG
 Command can not be scheduled

e_spinoff()
 NG
 Command can not be spunoff

e_ssys(x,opt)
 ?E
 INVALID SUBSYSTEM: <x>[; <opt>]
```

```
e_stderr()
  NG
ERROR OUTPUT OF PROGRAM MUST BE TO A TERMINAL

e_stdin()
  NG
INPUT TO PROGRAM MUST BE FROM A TERMINAL

e_stdout()
  NG
NORMAL OUTPUT OF PROGRAM MUST BE TO A TERMINAL
e_syntax(s1,s2,....,0)
 PROPER FORMAT: <s1>
                <s2>
              .

e_uperm(x,y,opt)
 ?E
 <x> RESTRICTED TO <y>[; <opt>]
```

The e_arb() routine is intended to output reflexive errors of an arbitrary format as described by the strings pointed to by s2,s3,... It is not intended that this routine be used in lieu of other reflexive routines which satisfy a specific need. Note that s1 is to be used as an acknowledgement string. That is, two spaces will be prepended to s1 and a newline will be appended to s1. The string will then be written out before continuing with the remaining arguments. Following the output of s1, the routine will output the following strings on the same line unless a character count of 75 is exceeded or an imbedded newline is encountered. The routine will attempt to break a line between specified strings or at an imbedded space char. Note that the routine will place a space between each string.

FILES

LIBRARY

/lib/libl.a

DIAGNOSTICS

Reports as above. Returns a 0 if successful and a -1 in case of error.

BUGS

NAME

e_output - output a stored OM

SYNOPSIS

```
#include <errfct.h>
e_output()
e_foutput(fildes)
   int fildes;
e_wrapup(inhflag)
   int inhflag;
```

DESCRIPTION

A stored OM (See e_syscall(3L), e_splerr(3L), e_intern(3L), e_form(3L)) is output. E_output uses sccerr(3) for outputting, unless the "primative report function" is set up by e_setup(3L) or e_setprim; e_foutput writes the OM to the file corresponding to the file descriptor fildes. (Handy for keeping your own log files.) E_wrapup checks the inhibits flag (See e_syscall(3L)) against the errno variable (See intro(2)) before outputting the OM. E_wrapup also calls the "trap function" if set up by e_settrap or e_setup(3L) after outputting the OM.

Outputting an OM does not erase it from storage and so an OM may be output sequentially by e_output and e_foutput.

SEE ALSO

e_syscall(3L), e_setup(3L)

DIAGNOSTICS

none

LIBRARY

/lib/lib1.a

NAME

e_reflex -- Input Message error reporting

SYNOPSIS

```
e_already(x,opt)  char *x, *opt;

e_arb(s1,s2,.....$n,0)  char *s1, *s2, ....,*$n;

e_chl(x,opt)  char *x, *opt;

e_exkw(x,opt)  char *x, *opt;

e_incd(x,y,opt)  char *x, *y, *opt;

e_inckw(x,y,opt)  char *x, *y, *opt;

e_inperr(opt)  char *opt;

e_invc(x,opt)  char *x, *opt;

e_invd(x,y,opt)  char *x, *y, *opt;

e_invkw(x,opt)  char *x, *opt;

e_ip()

e_kw(s1,s2, .....,0)  char *s1, *s2, ..

e_loc(x,opt)  char *x, *opt;

e_lperm(x,opt)  char *x, *opt;

e_misd(x,opt)  char *x, *opt;

e_miskw(x,opt)  char *x, *opt;

e_ng(s1,s2, .....,0)  char *s1, *s2, ..

e_ofc(x,opt)  char *x, *opt;

e_ok()

e_perm(x,opt)  char *x, *opt;

e_pf()

e_punct(x,y,opt)  char *x, *y, *opt;

e_rgerr(x,opt)  char *x, *opt;

e_rlbsy(opt)  char *opt;

e_rlovld(opt)  char *opt;
```

```
e_ssys(x,opt)  char *x, *opt;

e_stderr()

e_stdin()

e_stdout()

e_syntax(s1,s2, .....,0)  char *s1, *s2, ..

e_uperm(x,y,opt)  char *x, *y, *opt;
```

DESCRIPTION

Each function (except e_ok , e_ip , and e_pf) is provided as a standard method of responding to reflexive errors in an Input Message (IM) or in a prompted user response error.

e_ok , e_ip , and e_pf are provided for the generation of common, high usage non-error messages.

Each function is listed below with the error message format. Note, however, that the message acknowlegements (OK, NG, etc.) are not preceeded by a newline and will be outputted on the same line as the input command. Some functions require arguments "x" and "y" of type (char *). If non-zero, the string is inserted in the message where indicated by <x> and <y>. Some functions require an argument "opt" of type (char *). If non-zero, the string contained in square brackets will be outputted in the formats below.

Those functions accepting an arbitrary number of arguments "s1", "s2",... of type (char *) require that the last argument be zero.

Each of the routines produces the following output which is directed to file descriptor 2.

```
e_already(x,opt)
 NG
 ALREADY <x> [; <opt>]

e_arb(s1,s2,.....,0)
  s1
s2 s3 .....

e_chl(x,opt)
 ?E
 INVALID CHL: <x> [; <opt>]

e_exkw(x,opt)
 ?E
 EXTRA KEYWORD <x> [; <opt>]
```

```
e_incd(x,y,opt)
 ?E
 INCONSISTENT DATA <x>[, WITH <y>][; <opt>]

e_inckw(x,y,opt)
 ?E
 INCONSISTENT KEYWORD <x>[, WITH <y>][; <opt>]

e_inperr(opt)
 ?E
 INPUT ERROR [; <opt>]

e_invc(x,opt)
 ?E
 INVALID CHARACTER <x>[; <opt>]

e_invd(x,y,opt)
 ?E
 INVALID DATA <x> [FOR KEYWORD <y>][; <opt>]

e_invkw(x,opt)
 ?E
 INVALID KEYWORD: <x>[; <opt>]

e_ip()
 IP

e_kw(s1,s2,....,0)
 VALID KEYWORDS: <s1>
           <s2>
             .
             .

e_loc(x,opt)
 ?E
 INVALID LOCATION: <x>[; <opt>]

e_lperm(x,opt)
 ?E
 <x> NOT IN THIS LOCATION[; <opt>]

e_misd(x,opt)
 ?E
 MISSING DATA [FOR KEYWORD <x>][; <opt>]

e_miskw(x,opt)
 ?E
 MISSING KEYWORD <x>[; <opt>]

e_ng(s1,s2,...,0)
 NG
 <s1> <s2> ....
```

```
e_ofc(x,opt)
 ?E
 INVALID OFC: <x>[; <opt>]

e_ok()
 OK

e_perm(x,opt)
 ?E
 USE OF <x> NOT PERMITTED [; <opt>]

e_pf()
 PF

e_punct(x,y,opt)
 ?E
 INVALID PUNCTUATION; '<x>' SHOULD BE '<y>' [; <opt>]

e_rgerr(x,opt)
 ?E
 RANGE ERROR IN <x>[; <opt>]

e_rlbsy(opt)
 RL
 PROGRAM BUSY[; <opt>]

e_rlovld(opt)
 RL
 SYSTEM OVLD[; <opt>]

e_ssys(x,opt)
 ?E
 INVALID SUBSYSTEM: <x>[; <opt>]

e_stderr()
  NG
ERROR OUTPUT OF PROGRAM MUST BE TO A TERMINAL

e_stdin()
  NG
INPUT TO PROGRAM MUST BE FROM A TERMINAL

e_stdout()
  NG
NORMAL OUTPUT OF PROGRAM MUST BE TO A TERMINAL
e_syntax(s1,s2,....,0)
 PROPER FORMAT: <s1>
                <s2>

e_uperm(x,y,opt)
 ?E
 <x> RESTRICTED TO <y>[; <opt>]
```

The e_arb() routine is intended to output reflexive errors of an arbitrary format as described by the strings pointed to by s2,s3,... It is not intended that this routine be used in lieu of other reflexive routines which satisfy a specific need. Note that s1 is to be used as an acknowledgement string. That is, two spaces will be prepended to s1 and a newline will be appended to s1. The string will then be written out before continuing with the remaining arguments. Following the output of s1, the routine will output the following strings on the same line unless a character count of 75 is exceeded or an imbedded newline is encountered. The routine will attempt to break a line between specified strings or at an imbedded space char. Note that the routine will place a space between each string.

FILES

LIBRARY

/lib/lib1.a

DIAGNOSTICS

Reports as above. Returns a 0 if successful and a -1 in case of error.

BUGS

NAME

e_savename - save filename for standard I/O reporting

SYNOPSIS

```
e_savename (name, fildes)
     char *name;
     int fildes;
```

DESCRIPTION

Save a pointer to a file name for use in generating OM's for standard I/O calls on the stream associated with the file descriptor fildes. Normally fildes would be obtained from fileno (stream). This routine is called automatically by e_fopen(3) and e_freopen(3). It would be explicitly called by application programs only if the saved name is inappropriate or if the file descriptor is opened by some other means.

SEE ALSO

e_stdio(3L)

DIAGNOSTICS

none

BUGS

Only the pointer is saved, not the whole string.

LIBRARY

/lib/lib1.a

NAME

e_setup - Set up OM (Output Message) generating parameters

SYNOPSIS

```
#include <errfct.h>
char *e_setname (program_name)
   char *program_name;

char *e_setcode (errcode)
   char *errcode;

e_setlvl (almlvl)
   int almlvl;

int (e_setprim (prim_report_func))()
   int (*prim_report_func)();

int (*e_setglb())()

e_setrep (repeat_time)
   int repeat_time;

int (*e_settrap (trap_function))()
   int (*trap_function)();

e_setup (program_name, almlvl, repeat_time, prim,
     trap_function, errcode)
   int prim;
```

DESCRIPTION

Set parameters to be used by OM-generating functions such as e_syscall(3), e_stdio(3). E_setname **MUST** be used. Program_name points to a string containing the program name as it should be seen by the field (eg, "SCHEDULER" or "OP:MEAS").

E_setcode sets up a "standard error code" (see sccerr(3)) for the program. This will be used by e_splerr(3) and e_form(3). Errcode should point to a three letter (upper case) string.

E_setlvl sets up the alarm level for all OM's, if other than "minor" is desired. Almlvl should be one of the following define symbols: **LVLMINOR**, **LVLMAJOR**, **LVLCRIT**, or **LVLNONE**.

E_setprim sets up the "primitive report function". This is used to output OM's. If none is set, sccerr(3) is used; if one is set, it is called with arguments consistent with glberr(3). e_setglb, with a non_zero argument is equivalent to e_setprim(glberr).

E_settrap sets up a "trap function", called after an OM is output (except with output by e_output or e_foutput). It is called with a non_zero argument. If no trap function is set up, none is called.

Each of these routines returns the previous value of it parameter. E setup performs the functions of all of the above in one fell swoop. Exception: for the fourth argument (prim) if glb is equal to one, e setglb is simulated, if any other non-zero value is given, a call to e setprim is simulated. Any zero arguments to e setup cause the corresponding parameter to be unmodified.

LIBRARY

/lib/lib1.a

NAME

e_splerr - Generates "special" output message (OM)

SYNOPSIS

```
#include <errfct.h>
e_splerr (errnum,msg,inhflag)
   char *errnum;
   char *msg;
   int inhflag;
```

DESCRIPTION

An OM is created to describe a "special" No.2 SCCS error (ie, an error that needs to be described to the user in a separate OM Manual entry). The errnum argument specifies the error number of the OM; it should point to a three-digit numerical string. The error code of the OM is picked up from the standard error code for the process, as set by e setcode or e setup(3L).

The msg argument specifies the text field of the OM: it should point to a null terminated (upper case) string.

The inhflag argument is the standard error system inhibit flag (See e syscall(3L)). In this case only the values **ALLERR** (output the OM) or **NOERR** (just store away the OM) are meaningful.

LIBRARY

/lib/lib1.a

NAME

e_stdio - error generating versions of standard I/O routines

SYNOPSIS

```
#include <stdio.h>
#include <errfct.h>

char * e_cuserid(s, inhflag)
    char *s;
int  e_fclose(stream, inhflag)
    FILE *stream;
FILE * e_fdopen(fildes, type, inhflag)
    int fildes;
    char *type;
int  e_fflush(stream, inhflag)
    FILE *stream;
int  e_fgetc(stream, inhflag)
    FILE *stream;
char * e_fgets(s, n, stream, inhflag)
    char *s;
    int n;
    FILE *stream;
FILE * e_fopen(filename, type, inhflag)
    char *filename;
    char *type;
int  e_fprintf(stream, format, inhflag, arg1, arg2, ... arg10)
    FILE *stream;
    char *format;
    int arg1;
    /*  etc. -- NB only 10 args are effective */
int  e_fputc(c, stream, inhflag)
    int c;
    FILE *stream;
int  e_fputs(s, stream, inhflag)
    int s;
    FILE *stream;
int  e_fputw(w, stream, inhflag)
    int w;
    FILE *stream;
int  e_fread(ptr, siz, nitems, stream, inhflag)
    char *ptr;
    int siz;
    int nitems;
    FILE *stream;
FILE * e_freopen(filename, type, stream, inhflag)
    char *filename;
    char *type;
    FILE *stream;
int  e_fscanf(stream, format, inhflag, ptr1, ptr2, ... ptr10)
    FILE *stream;
    char *format;
    int *ptr1;
    /*  etc. -- NB only 10 args are effective */
```

```
int  e_fseek(stream, offset, ptrname, inhflag)
    FILE *stream;
    long offset;
    int ptrname;
long  e_ftell(stream, inhflag)
    FILE *stream;
int  e_fwrite(ptr, siz, nitems, stream, inhflag)
    char *ptr;
    int siz;
    int nitems;
    FILE *stream;
int  e_getc(stream, inhflag)
    FILE *stream;
int  e_getchar( inhflag)
char * e_gets(s, inhflag)
    char *s;
int  e_getw(stream, inhflag)
    FILE *stream;
int  e_pclose(stream, inhflag)
    FILE *stream;
FILE * e_popen(command, type, inhflag)
    char *command;
    char *type;
int  e_printf(format, inhflag, arg1, arg2, ... arg10)
    char *format;
    int arg1;
    /*  etc. -- NB only 10 args are effective */
int  e_putc(c, stream, inhflag)
    int c;
    FILE *stream;
int  e_putchar(c, inhflag)
    int c;
int  e_puts(s, inhflag)
    int s;
int  e_rewind(stream, inhflag)
    FILE *stream;
int  e_scanf(format, inhflag, ptr1, ptr2, ... ptr10)
    char *format;
    int *ptr1;
    /*  etc. -- NB only 10 args are effective */
int  e_setbuf(stream, buf, inhflag)
    FILE *stream;
    char *buf;
int  e_system(string, inhflag)
    char *string;
int  e_ungetc(c, stream, inhflag)
    int c;
    FILE *stream;
```

DESCRIPTION

These routines are analagous to those described in e_syscall(3L). The corresponding standard I/O routine - Section 3S - is called and its return value is in turn returned (like that?). See e_syscall(3L) for details of how errors are processed.

Note that no extra "name" arguments are required as in some e_syscall(3L) routines. Instead the file name passed to e_open or e_reopen is remembered and stdin, stdout and stderr are special cased. If a file is opened be other means or if you don't like the saved name, it may be changed by calling e_savename(3L).

LIBRARY

/lib/lib1.a

SEE ALSO

e_syscall(3L) and the sections referenced there
e_savename(3L)

DIAGNOSTICS

Same as corresponding routines in Section 3S.

BUGS

Attempts to use e_fprintf (and possibly a few others) with a stream opened for reading will not be detected since the standard I/O routine leaves no trace of such an error.

E_printf, e_fprintf, e_scanf and e_fscanf are limited to 10 arguments besides the format, stream, and innflag arguments.

NAME

e_syscall - error generating versions of system call routines

SYNOPSIS

```
#include <errfct.h>

int  e_access(name, mode, inhflag)
     char *name;
     int mode;
int  e_acct(name, inhflag)
     char *name;
int  e_attach(sub, gp, inhflag)
     int sub;
     int gp;
int  e_block(sema, inhflag)
     int sema;
int  e_brk(addr, inhflag)
     char *addr;
int  e_chan(gr, inhflag)
     int gr;
int  e_chdir(dirname, inhflag)
     char *dirname;
int  e_chmod(name, mode, inhflag)
     char *name;
     int mode;
int  e_chown(name, owner, group, inhflag)
     char *name;
     int owner;
     int group;
int  e_chroot(dirname, inhflag)
     char *dirname;
int  e_close(fildes, name, inhflag)
     int fildes;
int  e_connect(fd, ch, side, inhflag)
     int fd;
     int ch;
     int side;
int  e_creat(name, mode, inhflag)
     char *name;
     int mode;
int  e_csignal(index, gp, sig, inhflag)
     int index;
     int gp;
     int sig;
int  e_detach(sub, gp, inhflag)
     int sub;
     int gp;
int  e_dismaus(vaddr, inhflag)
     char *vaddr;
int  e_dup(fildes, name, inhflag)
     int fildes;
     char *name;
char * e_enabmaus(mausdes, inhflag)
```

```
        int mausdes;
    int e_errlog(flag, inhflag)  /* SC5 only */
        int flag;
    int e_execl(name, arg0, arg1 , ... 0, inhflag)
        char *name;
        char *arg0;
    int e_execle(name, arg0, arg1, ... 0, envp, inhflag)
        char *name;
        char *arg0;
        char *envp[];
    int e_execlp(name, arg0, arg1, ... 0, inhflag)
        char *name;
        char *arg0;
    int e_execv(name, argv, inhflag)
        char *name;
        char *argv[];
    int e_execve(name, argv, envp, inhflag)
        char *name;
        char *argv[];
        char *envp[];
    int e_execvp(name, argv, inhflag)
        char *name;
        char *argv[];
    int e_extract(sub, ch, side, inhflag)
        int sub;
        int ch;
        int side;
    int e_fcntl( fildes, request, argument, name, inhflag)
        int fildes;
        int request;
        int argument;
        char *name;
    int e_fork( inhflag)
    int e_freemaus(mausdes, inhflag)
        int mausdes;
    int e_fstat(fildes, buf, name, inhflag)
        int fildes;
        struct stat *buf;
        char *name;
    int e_getmaus(name, mode, inhflag)
        char *name;
        int mode;
    int e_gtty(fildes, arg, name, inhflag)      /* SC5 only*/
        int fildes;
        struct SGBUF *arg;
        char *name;
    int e_ioctl(fildes, request, argp, name, inhflag)
        char *fildes;
        int request;
        struct sgttyb *argp;
        char *name;
    int e_join(fd, xd, inhflag)
        int fd;
```

```
        int xd;
int     e_kill(pid, sig, inhflag)
        int pid;
        int sig;
int     e_link(name1, name2, inhflag)
        char *name1;
        char *name2;
int     e_lock(sema, inhflag)
        int sema;
long    e_lseek( fildes, offset, whence, name, inhflag)
        int fildes;
        long offset;
        int whence;
        char *name;
int     e_mknod(name, mode, addr, inhflag)
        char *name;
        int mode;
        int addr;
int     e_mount(special, name, rwflag, inhflag)
        char *special;
        char *name;
        int rwflag;
int     e_mpx(name, mode, inhflag)
        char *name;
        int mode;
int     e_msgdisab( inhflag)
int     e_msgenab( inhflag)
int     e_nice(priority, inhflag)
        int priority;
int     e_open(name, mode, inhflag)
        char *name;
        int mode;
int     e_p(sema, inhflag)
        int sema;
int     e_pause( inhflag)
int     e_pipe(fildes, inhflag)
        int *fildes;
int     e_post(sema, inhflag)
        int sema;
int     e_ptrace(request, pid, addr, data, inhflag)
        int request;
        int pid;
        int addr;
        int data;
int     e_rdsem(sema, inhflag)
        int sema;
int     e_read(fildes, buffer, nbytes, name, inhflag)
        int fildes;
        char *buffer;
        int nbytes;
        char *name;
int     e_recv(buf, size, type, inhflag)
        char *buf;
```

```
        int size;
        char *type;
int     e_recvw(buf, size, type, inhflag)
        char *buf;
        int size;
        char *type;
char *  e_sbrk(incr, inhflag)
        int *incr;
int     e_send(buf, size, topid, type, inhflag)
        char *buf;
        int size;
        int topid;
        int type;
int     e_sendw(buf, size, topid, type, inhflag)
        char *buf;
        int size;
        int topid;
        int type;
int     e_setgid(gid, inhflag)
        int gid;
int     e_setpgrp(pid, inhflag)
        int pid;
int     e_setsem(sema, value, inhflag)
        int sema;
        int value;
int     e_setuid(uid, inhflag)
        int uid;
int     (*e_signal(sig, func, inhflag) ) ()
        int sig;
        int (*func)();
int     e_stat(name, buf, inhflag)
        char *name;
        struct stat *buf;
int     e_stime(tp, inhflag)
        long *tp;
int     e_stty(fildes, arg, name, inhflag)
        int fildes;
        struct SGBUF *arg;
        char *name;
char *  e_switmaus(mausdes, vaddr, inhflag)
        int mausdes;
        char *vaddr;
int     e_sync( inhflag)
long    e_tell(fildes, name, inhflag)
        int fildes;
        char *name;
int     e_test(sema, inhflag)
        int sema;
long    e_time(tloc, inhflag)
        long *tloc;
long    e_times(buffer, inhflag)
        struct tbuffer *buffer;
int     e_tlock(sema, inhflag)
```

```
        int sema;
daddr_t  e_ulimit(newlimit, inhflag)
        daddr_t newlimit;
int     e_umask(mask, inhflag)
        int mask;
int     e_umount(special, inhflag)
        char *special;
int     e_unlink(name, inhflag)
        char *name;
int     e_unlock(sema, inhflag)
        int sema;
int     e_utime(file, times, inhflag)
        char *file;
        struct utimbuf times;
int     e_v(sema, inhflag)
        int sema;
int     e_wait(wait, status, inhflag)
        int wait;
        int *status;
int     e_write(fildes, buffer, nbytes, name, inhflag)
        int fildes;
        char *buffer;
        int nbytes;
        char *name;
int     e_xread(fildes, buffer, nbytes, name, inhflag)
        int fildes;
        char *buffer;
        int nbytes;
        char *name;
```

DESCRIPTION

Each routine calls the corresponding system call routine in Section 2, and returns the same value returned by the system call. If an error is detected:

1) If the error is overload related (eg, inability to open a file because the inode table is full) and if the program has previously called e_setrep (or e_setup(3L) with appropriate arguments - see e_setup(3L)), then the system call is repeated for the time specified in the e_setrep call or until the error clears.

2) If another type of error is detected or if the overload error is not resolved, then an SCCS output message (OM) is formatted and stored away.

3) If an OM is created and if the "error class" (as determined by the errno variable - see Intro(2)) is not "inhibited", then the OM is outputted by means of sccerr or glberr. The inhibiting is controlled by the "inhflag" argument. Its value may be any of the following defines (<errfct.h>) or the logical ORing of two or more of them:

Define	Error Class Inhibited	Errno
INHPERM	Permission (su or owner)	EPERM
INHNOENT	File not found	ENOENT
INHINTR	System call interrupted	EINTR
INHNXIO	Non existing special file	ENXIO
INHNOEXEL	No execute permission	ENOEXEC
INHAGAIN	Process table overflow	EAGAIN
INHACCES	File access permission	EACCESS
INHNOTDIR	File should be a directory	ENOTDIR
INHNFILE	File or inode table overflow	ENFILE
INHFBIG	File too big	EFBIG
INHZBIG	Argument list too big	EZBIG
INHNOSPC	Out of disk space	ENOSPC
NOERR	Inhibit for all	-

If no inhibiting is desired, use **ALLERR** for the inhibit flag.

Numerous options are available - see e_setup(3L).

The routine e_setname(3L) or e_setup(3L) must be called prior to these routines to set up the program name field of the OM.

4) OM's stored away may be modified and then output - see e_new(3L) and e_output(3L).

Note that several routines have an extra "name" argument. This should be a pointer to the name of the file being operated upon or zero if the name is not known.

LIBRARY

/lib/lib1.a

SEE ALSO

e_stdio(3L) e_setup(3L) e_new(3L) e_output(3L) intro(2)

DIAGNOSTICS

Same as corresponding routines in Section 2.

NAME

fgid -- find group id

SYNOPSIS

```
fgid(group)
char *group;
```

DESCRIPTION

Fgid looks in the /etc/group file for a given group name and returns the numerical group id if found, or,

-1 - if it cannot find group

-2 - if it cannot open /etc/group

The standard I/O subroutine **getgrnam(3)** is used which uses **malloc(3)**

LIBRARY

/lib/lib1.a

FILES

/etc/group

SEE ALSO

fgname(3), getgrnam(3)

DIAGNOSTICS

BUGS

NAME

isdemand,issched,isspinoff,isbackgrd - execution environment

SYNOPSIS

isdemand()

issched()

isspinoff()

isbackgrd()

DESCRIPTION

All routines reference the environment variable **EX_MODE** to retrieve the execution mode. In all cases, the returned value is either 0 to represent a 'no' condition or 1 to represent a 'yes' condition.

isdemand indicates if execution initiated from a terminal.

issched indicates if execution initiated by the SCCS Scheduler.

isspinoff indicates if background execution initiated from a terminal.

isbackgrd indicates if execution is either of the scheduled or spinoff case.

LIBRARY

/lib/libl.a

NAME

fgname -- find group name

SYNOPSIS

```
char *fgname(gid)
int gid;
```

DESCRIPTION

fgname accepts gid, an integer group id value, searches /etc/group file for the corresponding group name and returns:

-1 - if the group id is not found.
-2 - if fgname had trouble opening /etc/group file.
<string pointer to group name> - if the group id is found.

getgrgrid(3) is used which uses **malloc(3)**

LIBRARY

/lib/lib1.a

FILES

/etc/group

SEE ALSO

fgid(3), getgrgrid(3)

DIAGNOSTICS

BUGS

NAME

fmterr -- SCC error formatting routine

SYNOPSIS

```
fmterr(fdes,fmt,arg1,arg2,...,argn)
int fdes;
char *fmt,*arg1,*arg2,...,*argn;
```

DESCRIPTION

This subroutine is a special string formatter used by the error handling routines, glberr and sccerr, to print the various elements of an SCC error message.

Fmterr has the following arguments:

fdes is either the file descriptor of the device on which the message is to be printed. For sccerr, fdes equals 2; for glberr, fdes is the file descriptor for the system teletypewriter or the negative of the process id of thte system spooler.

fmt is a special format string which controls the printing of the error message. Default characters from this string are simply outputted to the specified device. The following characters have special meanings.

\0 terminate output.

\1 go to the next argument and output one character.

\2 output one character from the current argument.

\3 output the next argument as a string.

\4 append an End of Msg (031) char, to the msg.

arg1,arg2,...,argn
a variable number of arguments, each being a pointer to a character string.

LIBRARY

/lib/lib1.a

SEE ALSO

glberr(3), sccerr(3)

DIAGNOSTICS

NAME

fpid -- get process id

SYNOPSIS

```
int fpid(line)
char *line;
```

DESCRIPTION

FUNCTION:

The fpid subroutine searches the /etc/utmp file for the process line name specified by the argument line. The routine returns the process id (pid) of the line if the search was successful.

ARGUMENTS:

line -- a pointer to a character string containing the process line name.

RETURNS:

pid -- process id of the given line name
0 -- found the line name, but process is probably dead
-1 -- given line name could not be found
-2 -- system problem; check errno for exact system error

FILES

/etc/utmp --- open, read, and close it.

LIBRARY

/lib/lib1.a

NAME

freesort -- variable length record sort

fixedsort -- fixed length record sort

SYNOPSIS

```
freesort (argc, argv, comp)
fixedsort (argc, argv, comp)
int argc;
char **argv;

int (*comp)(pp1,pp2);
char **pp1, **pp2;
extern char DELIM = '\n'; /* for freesort() only */
extern char RCDSIZ = 0; /* for fixedsort() only */
extern int cmpflg;

extern struct {
     int e_code;
     char e_msg[50];
} errsort;
```

DESCRIPTION

freesort() and fixedsort() are provided as alternatives to the standard UNIX sort for those instances that the standard comparison routine is not appropriate. These routines use standard I/O. For "large" sorting jobs, freesort() and fixedsort() may use up to 10 streams above and beyond that of the routine that calls them; if 10 file descriptors are not available, then the subroutine will fail.

The comparison routine should return a value greater than 0 when the record corresponding to the first argument is "greater than" (i.e. should precede in the output) the record corresponding to the second argument. Analogously, a negative value should be return by the comparison routine when the first record is "less than" (i.e. should follow in the output) the second record. When the comparison routine returns the value 0, it means that the routine does not care which is first.

By default, the algorithm is not stable (i.e. it does not preserve the order of records with identical sort codes. However, options are available to either preserve or reverse original order which are rather efficient for most applications.

freesort() and fixedsort() work by allocating almost all of the available memory, repeatedly filling that core with records which they sort and dump to disc, and then they merge the disc files (if necessary). Therefore, these subroutines may not work well unless lots of unallocated memory is available to this routine. However, with separated I&D space, it should be very seldom when the calling routine takes up so much of the available memory that these routines cannot run efficiently compared to the costs of

executing a smaller main which executes freesort() or fixedsort().

Upon exiting, even in the case of an error, all streams are closed and the allocated memory is returned to the system. freesort() and fixedsort() can be called repeatedly from the same routine as a part of a larger algorithm (the old versions could not). freesort() and fixedsort() catch interrupts, hangups, and quits in order to clean up the temporary files which they make. They return to main the value 6 after they have cleaned up in order to give the main routine the same opportunity to clean up. In all cases, when freesort() and fixedsort() return to main, the process is in a mode to ignore interrupts, hangups and quits; thus the calling routine may wish to reset those signals.

freesort() tosses trivial records (those which only contain the delimiter). For each input file which does not end with a delimiter, freesort() behaves as if a delimiter were added to the input file. However, if fixedsort() encounters an odd part of a record at the end of any input file, that partial record is discarded.

The first element of argv is ignored. The remaining argument strings will have the following interpretation:

-m merge only. All input files are assumed to be sorted.

-u output records with unique sort keys only.

-o the next argument is taken to be the output file. If none are given or the output file is "-", then stdout is assumed.

-s<char> or -s<size>
For freesort(), <char> is the delimiter. The delimiter defaults to '\n'. For fixedsort(), <size> is the record size. It defaults to zero, which if left there, causes an error.

-c<value> The external variable, cmpflg, which may be used as a flag by the comparison routine is set to the value of the ascii string, <value>.

-t<threshold> After partitioning the records into sets of identical sort keys, only the sets with <threshold> or more records are output.

-l<size> For freesort(), the limit of the record size. Records which exceed this size are truncated to <size> bytes (including the delimiter).

-T<string> For freesort() when a record is truncated, <string>

will be placed at the end of the record. If the "-l<size>" is not specified, then freesort() chooses the maximum size so that at least three records can fit into the allocated data space. If neither the "-T" or "-l" options are used, then freesort() will return abnormally if it encounters a record which is too large to handle.

<filename> Arguments which do not begin with "-" or follow an "-o" argument are assumed to be input files. No more than thirty input files are allowed. If there are no input file arguments, then stdin is assumed.

-d For freesort(), a delimiter will be placed as the first character in the output stream so that all records are "surrounded" by delimiters.

-P In the output, preserve the order of the records on input if they have identical sort codes.

-R In the output, reverse the order of the records on input if they have identical sort codes.

LIBRARY

/lib/lib1.a

DIAGNOSTICS

These routines return 0 for normal execution. A variety of non-zero returns occur when the subroutine does not terminate normally. When that occurs, the return value will also be written in errsort.e code and an error message will be written in errsort.e msg. The error message may help the calling routine construct an error message for the user. No message is written when the return value is 0 (normal) or 6 (interrruption by interrupt, hangup or quit signal). The structure of errsort, named ERRSORT, is found in "/compool/sorterr.h".

NAME

fuid -- find user id in passwd file

SYNOPSIS

```
fuid(user)
char *user;
```

DESCRIPTION

Fuid looks in the /etc/passwd file for a given user name and returns the numerical user id if found or,

-1 - if it cannot find user

-2 - if it cannot open /etc/passwd

getpwnam(3) is called, which calls **malloc(3)**.

LIBRARY

/lib/lib1.a

FILES

/etc/passwd

SEE ALSO

getpw(3), getpwnam(3)

DIAGNOSTICS

BUGS

NAME

gen_list - extract next generic-issue message from issue file

SYNOPSIS

```
#include <issfil.h>

char *gen_list(fd)
int fd;
```

DESCRIPTION

Gen list should be used by those routines that need to generate a list of supported generics for a particular office type. This subroutine extracts the next generic record and its associated issue records (generic-issue message) from the indicated issue file and returns the starting address of the generic record to the calling routine. If the next generic-issue message can not be found or an EOF is detected, then the value **GLR_NME** is returned to the calling routine. If an error is detected, a negative value is returned as discussed below.

The user should note that the generic record is first copied to a static global character buffer and terminated with a null. The address of this static global character buffer is then returned to the calling routine. Data should be extracted from the record via the structure members defined in the header file issfil.h, however, prior to making a call to gen name(3L), get gen(3L), get iss(3L), or iss list(3L). These subroutines also use the same static global character buffer and a call to one of them would probably destroy the generic record extracted by this subroutine.

The argument fd is a file descriptor associated with an opened issue file.

FILES

/usr/include/issfil.h which specifies the structure of a generic record and an issue record and defines valid function codes and return codes for this subroutine.

LIBRARY

/lib/lib1.a

SEE ALSO

get_gen(3L), get_iss(3L), gen_name(3L), iss_list(3L), e_output(3L)

DIAGNOSTICS

If this subroutine detects an error, an Output Message (OM) is generated by one of the standard OM generation subroutines, but not printed. The value **GLR_ERR** is returned to the calling routine. If the calling routine wishes to print the stored OM, it may call one of the standard OM outputting subroutines, such as e output(3L).

NAME

gen_name - extract indicated generic name from issue file

SYNOPSIS

```
#include <issfil.h>

gen_name(func, ofcnam, genid, name)
int func;
char *ofcnam;
char *genid;
char *name;
```

DESCRIPTION

Gen name opens an appropriate issue file that is pertinent to the specified office. It then searches this file to see if a generic-issue message corresponding to the specified generic ID exists. If the desired generic-issue message does exist, then either the official generic name or generic slang name is extracted from the generic record and copied to name and the value **GNR_EF** is returned to the calling routine. If the desired generic-issue message does not exist, the value **GNR_ENF** is returned. If an error is detected, a negative value is returned as discussed below.

The user should note that the generic record is temporarily copied to the same static global character buffer that is used by gen list(3L), get gen(3L), get iss(3L), and iss list(3L). Thus, the data record extracted by one of these subroutines would be destroyed by the call to gen name.

The argument func identifies which generic name is to be extracted from the generic record. Valid values for this argument are:

GNF_GNAM Extract generic name.

GNF_SLANG Extract generic slang name.

The argument ofcnam is a null-terminated string that contains the office name.

The argument genid is a null-terminated string containing the generic ID that has been extracted from the oparm file in the ofcnam directory.

The argument name is the address of a character array to which the requested generic name is to be copied. The generic name will be null-terminated. Gen name assumes that this array has a declared size that is equal to **IF_GNAMSZ** + 1 or **IF_SLGSZ** + 1, depending upon which value of func has been specified.

FILES

/usr/include/issfil.h which specifies the structure of a generic record and defines valid function codes and return codes for this subroutine.

LIBRARY

/lib/lib1.a

SEE ALSO

get_gen(3L), get_iss(3L), gen_list(3L), iss_list(3L), e_output(3L)

DIAGNOSTICS

If this subroutine detects an error, an Output Message (OM) is generated by one of the standard OM generation subroutines, but not printed. The value **GNR_ERR** is returned to the calling routine. If the calling routine wishes to print the stored OM, it may call one of the standard OM outputting subroutines, such as e_output(3L).

BUGS

NAME

gen_rng - locate specified entry in generic range data file

SYNOPSIS

```
#include <gen_rng.h>

char *gen_rng(ofcnam,feature,featfun)
char *ofcnam;
char *feature;
char *featfun;
```

DESCRIPTION

Gen rng searches an appropriate generic range file to see if an entry corresponding to the specified feature and featfun exists. If the entry does not exist, the value **GRR_ENF** is returned. If the requested entry does exist, then the starting address of the entry is returned to the calling routine. If an error is detected, a negative value is returned as discussed below.

The user should note that the generic range record is terminated by a null and data should be extracted from the record via the structure members defined in the header file, gen rng.h.

The argument ofcnam is a null-terminated string that identifies the office for which the feature is being performed. The office name steers this subroutine to a particular /type?? directory wherein resides the generic range file that is to be used.

The argument feature is a null-terminated string that identifies which feature is to be performed. Examples are **rcb** for RC:BUILD and **sca** for Scheduled Common Analysis.

The argument featfun is a null-terminated string that identifies which one of the feature's functions is to be performed. For example, the feature **sca** has several functions, such as **spa** for Switched Path Analysis, **eca** for External Circuit Analysis, and **nca** for Network Controller Analysis. If a feature has only one function, then this argument may contain a null string.

FILES

/usr/include/gen rng.h which specifies the structure of a generic range file entry.

LIBRARY

/lib/lib1.a

SEE ALSO

e_output(3L)

DIAGNOSTICS

If this subroutine detects an error, an Output Message (OM) is generated by one of the standard OM generation subroutines, but not printed. The value **GRR_ERR** is returned to the calling

routine. If the calling routine wishes to print the stored OM, it may call one of the standard OM outputting subroutines, such as e_output(3L).

BUGS

NAME

get_gen - extract specified generic-issue message from issue file

SYNOPSIS

```
#include <issfil.h>

char *get_gen(func, fd, gen)
int func;
int fd;
char *gen;
```

DESCRIPTION

Get gen extracts the specified generic record and its associated issue records (generic-issue message) from the indicated issue file and returns the starting address of the generic record to the calling routine. If the desired generic-issue message does not exist, then the value **GGR_ENF** is returned. If an error is detected, a negative value is returned as discussed below.

The user should note that the generic record is first copied to a static global character buffer and terminated with a null. The address of this static global character buffer is then returned to the calling routine. Data should be extracted from the record via the structure members defined in the header file issfil.h, however, prior to making a call to gen list(3L), gen name(3L), get iss(3L), or iss list(3L). These subroutines also use the same static global character buffer and a call to one of them would probably destroy the generic record extracted by this subroutine.

The argument func identifies whether the generic name, generic slang name, or generic ID is to be used as the generic search key. Valid values for this argument are:

GGF_GNAM Use generic name as the generic search key.

GGF_SLANG Use generic slang name as the generic search key.

GGF_GID Use generic ID as the generic search key.

The argument fd is a file descriptor associated with an opened issue file.

The argument gen is a null-terminated string containing the generic search key. If the value of func is **GGF_GNAM**, **GGF_SLANG**, or **GGF_GID**, then this key should contain the official generic name, generic slang name, or generic ID, respectively.

FILES

/usr/include/issfil.h which specifies the structure of a generic record and defines valid function codes and return codes for this subroutine.

LIBRARY

/lib/lib1.a

SEE ALSO

get_iss(3L), gen_list(3L), gen_name(3L), iss_list(3L), e_output(3L)

DIAGNOSTICS

If this subroutine detects an error, an Output Message (OM) is generated by one of the standard OM generation subroutines, but not printed. The value **GGR_ERR** is returned to the calling routine. If the calling routine wishes to print the stored OM, it may call one of the standard OM outputting subroutines, such as e_output(3L).

BUGS

NAME

get_iss - locate specified issue record in issue data file

SYNOPSIS

```
#include <issfil.h>

char *get_iss(iss)
char *iss;
```

DESCRIPTION

Get iss locates the specified issue record in the generic-issue message that has been extracted from the issue file by a previous call to the library subroutine, get gen(3L). If the desired issue record does exist, then the starting address of the issue record is returned to the calling routine. If the desired issue record does not exist, then the value **GIR_ENF** is returned.

The user should note that the issue record is first copied to a static global character buffer and terminated with a null. The address of this static global character buffer is then returned to the calling routine. Data should be extracted from the record via the structure members defined in the header file issfil.h, however, prior to making a call to gen list(3L), gen name(3L), get gen(3L), or iss list(3L). These subroutines also use the same static global character buffer and a call to one of them would probably destroy the issue record extracted by this subroutine.

The argument iss is a null-terminated string that identifies the desired issue and point issue. It must be of the form **ii.pp** where **ii** identifies the issue, such as **3**, **6a**, or **10c**; the "." is a delimiter character, and **pp** identifies the point issue, such **1**, **2**, or **12**.

FILES

/usr/include/issfil.h which specifies the structure of an issue record and defines valid return codes for this subroutine.

LIBRARY

/lib/lib1.a

SEE ALSO

get_gen(3L), gen_list(3L), gen_name(3L), iss_list(3L)

DIAGNOSTICS

BUGS

NAME

getdfprm, setdfprm - default SCCS parameters (office, etc.)

SYNOPISIS

```
char *getdfprm(name)
   char *name;
char *setdfprm(name, value)
   char *name;
   char *value;
```

DESCRIPTION

getdfprm and setdfprm search the .dfltparm file in the current directory for a line of the form

name=oldvalue

If such a line is found, the string oldvalue is returned.

setdfprm, in addition, does the following:

1) If such a line is found, oldvalue is replaced by value,

2) If such a line is not found, the line

name=value

is added to the file and value is returned,

3) If the .dfltparm file does not exist, it is created, and the same line as in 2) is put in it.

NOTES

Lib3 is used by setdfprm.

The string returned by getdfprm or setdfprm is distroyed by another call to getdfprm or setdfprm.

FILES

.dfltparm
.dfltparm<pid> temporary file
/usr/include/aparam.h has the following defines:

```
#define DFLTPARM ".dfltparm"
#define DFLTOFC "OFFICE"
```

DIAGNOSTICS

getdfprm returns NULL if a default line for name is not found. setdfprm returns NULL if the length of value is greater than MAX-VALUE (currently 20). NULL is also returned if an open error (or link error, in the case of setdfprm) is encountered, in which case errno is set, and an "e_" error message is stored, that can be output by e_output(3E). An open error indication from getdfpram generally should be ignored, since this just means that

the .dfltparm doesn't exist yet.

SEE ALSO

getenv(3C), updofc(3E), lopen(3E)

LIBRARY

/lib/lib1.a

NAME

getds -- process requests for data communications equipment

SYNOPSIS

```
#include <dial.h>

struct dntbl *getds(baudrate, WATS_or_not, ans_mode_term)
char *baudrate;
short WATS_or_not, ans_mode_term;
```

DESCRIPTION

The purpose of this subroutine is to process all requests for data communications equipment . Given the desired baudrate and whether or not a WATS line is needed, this subroutine locates a free data set of the requested characteristics, allocates it to the requestor and indicates certain other characteristics about that data set (such as the associated multiplexor line and dn11).

It is the responsibility of the user to open and close the appropriate multiplexor line .

The requesting process is told to ingore the SIGPWR signal so that a periodic check routine may send that signal in order to check for the existence of the process.

Arguments:

baudrate -- "110","300", or "1200"
WATS_or_not -- 1 if WATS is needed, 0 otherwise
ans_mode_term -- 1 if calling terminal is answer-mode only

Return values:

It returns a pointer to a structure like dntbl (/usr/include/dial.h) which specifies the characteristics of the allocated equipment.

Errors are indicated as follows :

-1	No carrier, busy, or no answer.
-2	All equipment in use.
-3	Bad speed specification.
-4	Bad telephone number.
-5	Ioctl failure.
-6	Bad equipment resource table.
-7	No equipment exists - not specifyed in equipment resource table.

NOTE :

Users must release the equipment with a call to releaseds(3L).

FILES

/usr/include/dial.h /etc/d_dntable

LIBRARY
/lib/lib1.a

SEE ALSO
releaseds(3L), dial(3L)

NAME

getfld -- locate a specified field within a specified line of ASCII data

SYNOPSIS

```
#include <gtmhdr.h>

getfld(line,field,inbuf)
int line;
int field;
struct GMBUF *inbuf;
```

DESCRIPTION

Getfld breaks a specified line of input data into its respective fields, starting with field 0. Field separation characters are one or more tabs and/or blanks, a newline, an octal 212, or a null byte. The address of the requested field is returned in the structure variable, gm_fptr, and the value returned by getfld is the length of the field, in bytes. If an error is detected, a negative value is returned as discussed below.

The ASCII data buffer, which is a structure of type GMBUF, is declared and allocated by the calling routine. Before calling this subroutine, the calling routine must first fill the ASCII data buffer via the subroutine gtmsg(3L) or some other routine which performs a similar function.

The argument line is the number of the line in which the requested field is located. The range of values for line are:

0 <= line < **GM_MAX_LNS**

The argument field is the number of the field that is to be located. The range of values for field are:

0 <= field < max. fields for line

The argument inbuf is the address of a data buffer whose format is:

```
struct GMBUF
  {     int gm_fd;
        int gm_len;
        int gm_delim;
        int gm_lncnt;
        int gm_nchar;
        char *gm_lptr[GM_MAX_LNS];
        char *gm_fptr;
        char *gm_bufp;
        char *gm_bufe;
        char gm_buf[GM_BUFSIZ + 2];
  };
```

where